

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Smartphone-based Inertial Navigation System for Bicycles

Cristiano Alexandre Almeida Oliveira Rodrigues



Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Daniel Castro Silva (PhD)

Second Supervisor: Bruno Lage Aguiar (MSc)

July 31, 2015

Smartphone-based Inertial Navigation System for Bicycles

Cristiano Alexandre Almeida Oliveira Rodrigues

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: José Manuel de Magalhães Cruz (PhD)

External Examiner: Helena Rodrigues (PhD)

Supervisor: Daniel Castro Silva (PhD)

July 31, 2015

Resumo

Actualmente, em África, milhões de pessoas sofrem todos os dias com doenças, que se propagam facilmente devido às más condições de saúde e de vida existentes. "Syndromic Surveillance" é um projecto cujo objectivo passa por colectar dados no terreno sobre a existência destas doenças para conseguir fazer uma monitorização e previsão do mapa de doenças numa região Africana em concreto: a África subsariana. Os "Community Health Workers" são voluntários que viajam entre aldeias daquela região Africana, utilizando para isso bicicletas como meio de transporte. Transportando e utilizando os seus smartphones, colecionam dados georeferenciados relevantes para a monitorização feita pelo "Syndromic Surveillance". Os receptores de GPS integrados nestes dispositivos facilitam esta mesma georeferenciação, mas infelizmente, o sinal de GPS nem sempre está disponível por um variado número de factores, como as grandes elevações no terreno ou a falta de redes 3G para fornecerem dados AGPS.

Nesta dissertação, é proposta uma solução para contornar este problema utilizando um sistema de navegação inercial, o *BikeNav System*, que utiliza os sensores já existentes nos "smartphones" combinados com um odómetro externo para uma melhor precisão nas medições de velocidade e distância. Utilizando uma framework de testes especificamente criada do zero para o propósito de testar o sistema com dados previamente colectados através de voltas de bicicleta, as experiências efectuadas demonstraram que o sistema é capaz de estimar posições consistentemente dentro de uma área de 1000 m^2 à volta da posição real naquele momento.

Abstract

Currently, in Africa, millions of people suffer everyday from diseases, which propagate easily due to poor life and healthcare conditions. Syndromic Surveillance is a project which aims to collect data in order to monitor and predict diseases in a certain African geographical area. Community Health Workers are volunteers who travel by bicycle between villages, collecting georeferenced data with their smartphones. The GPS receivers embedded on these smartphones make localization easier; unfortunately, GPS signal is not always available, due to factors such as terrain elevations or the lack of 3G networks to provide AGPS data.

In this thesis, a solution is proposed through a modular and reusable inertial navigation system, the *BikeNav System*, which uses the already existing sensors in the smartphone, combined with an external odometer for better accuracy in speed and displacement readings. Using a comprehensive, created from scratch test framework for the specific purpose of testing the system with pre-collected data from bicycle rides, experiences show that the system is able to estimate positions consistently within an area of 1000 m^2 around the real position in that moment.

Acknowledgements

I would like to thank my family, my friends, and all of those who believed in me and supported me throughout these years.

I would also like to thank my supervisors Daniel Silva and Bruno Aguiar for always helping me when I needed, and Fraunhofer Portugal for giving me the opportunity to grow as a professional during these last months.

Cristiano Rodrigues

“There was a second supremely sweet moment of victory. As I made my way through the finish area, I passed the Cofidis team. Assorted members of the organization stood around, the men who I felt had left me for dead in a hospital room. ‘That was for you’, I said as I moved past them.”

Lance Armstrong

Contents

1	Introduction	1
1.1	Problem context and motivation	1
1.2	Goals	2
1.3	Document structure	3
2	Literature review	5
2.1	Useful Sensors for Localization	5
2.1.1	Sensors available in smartphones	5
2.1.2	Sensors not available in smartphones	10
2.1.3	Sensor fusion	11
2.2	Positioning techniques	17
2.2.1	Absolute positioning techniques	17
2.2.2	Global Navigation Satellite Systems	18
2.2.3	Map Matching	21
2.2.4	Relative positioning techniques	22
2.2.5	Inertial Navigation	22
2.3	Earth distances	24
2.3.1	Abstractions and representations	24
2.3.2	Haversine formula	25
3	Solution Architecture	27
3.1	Solution Requirements	27
3.2	Architecture Overview	27
3.3	BikeNav Library	29
3.3.1	Providers	31
3.3.2	Trackers	32
4	Data Collection and Results	39
4.1	Test Process Overview	39
4.2	Recorder Application	40
4.3	BikeNav Library Testing	42
4.3.1	Data loading and injection	42
4.3.2	Exportation of Results	44
4.4	Test Runs and Obtained Results	44
4.4.1	Odometer distance validation	44
4.4.2	Heading validation	46
4.4.3	Tracks with GPS cuts	50

CONTENTS

5	Conclusion	59
5.1	Final Remarks	59
5.2	Future Work	60
	References	61
A	Syndromic Surveillance Flyer	65

List of Figures

1.1	Diagram of the Syndromic Surveillance project.	2
2.1	Representation of the gyroscope architecture.	6
2.2	Representation of the accelerometer axis in smartphones.	6
2.3	The Earth's magnetic field.	7
2.4	Example of mercurial barometer montage.	8
2.5	An example of an odometer for bikes.	10
2.6	Google's self-driving car.	11
2.7	Representation of the body and global frame.	12
2.8	Gaussian (or normal) distribution curve.	14
2.9	Trilateration example.	18
2.10	Triangulation example using three beacons.	18
2.11	Orbits from each of the GNSSs' satellites.	19
2.12	NAVSTAR GPS satellite network representation.	19
2.13	Garmin Edge 810 GPS receiver.	21
2.14	Polar V650 GPS receiver.	21
2.15	Map building using RViz.	21
2.16	An example of an inertial navigation system architecture.	22
2.17	Actual positions (blue), calculated positions (red), calculated positions under noise (green).	23
2.18	Actual trajectory (blue), calculated trajectory (red), calculated trajectory under noise (green).	23
2.19	Geoid and ellipsoid representation models.	24
2.20	Geoid representation example.	24
2.21	Mercator projection of the Earth surface.	25
3.1	High-level overview of the solution architecture.	28
3.2	Overview of the BikeNav library system architecture.	29
3.3	Overview of Altitude Tracker's flow of information.	32
3.4	Overview of Gyroscope-based Heading Tracker's flow of information.	33
3.5	Representation of magnetometer vectors and calculated heading angles.	34
3.6	Overview of Magnetometer-based Heading Tracker's flow of information.	34
3.7	Overview of Revolution Tracker's flow of information.	35
3.8	Overview of Displacement Tracker's flow of information.	36
3.9	Overview of Location Tracker's flow of information.	36
3.10	Overview of Position Tracker's flow of information.	37
4.1	Overview of the iterative development process.	39
4.2	The Recorder application, used to collect data from bike rides.	40

LIST OF FIGURES

4.3	The speed sensors, mounted on the test bicycles.	41
4.4	High-level overview of the test framework structure.	42
4.5	Representation in Google Maps of the 'Recta de Casa' segment with most accumulated drift using gyroscope-based heading tracker.	47
4.6	Representation in Google Maps of the 'Recta de Casa' segment with least accumulated drift using gyroscope-based heading tracker.	47
4.7	Representation in Google Maps of the 'Recta Rua João Calisto (inverse)' segment with a large error using magnetometer-based heading tracker.	49
4.8	Representation in Google Maps of the 'Recta Rua João Calisto (inverse)' segment with a small error using magnetometer-based heading tracker.	49
4.9	Representation in Google Maps of track 1.	51
4.10	Representation in Google Maps of the first run of track 1 using magnetometer-based heading tracker.	52
4.11	Representation in Google Maps of the first run of track 1 using gyroscope-based heading tracker.	52
4.12	Representation in Google Maps of the second run of track 1 using magnetometer-based heading tracker.	53
4.13	Representation in Google Maps of the second run of track 1 using gyroscope-based heading tracker.	54
4.14	Representation in Google Maps of track 2.	55
4.15	Representation in Google Maps of the first run of track 2 using magnetometer-based heading tracker.	56
4.16	Representation in Google Maps of the first run of track 2 using gyroscope-based heading tracker.	57
4.17	Representation in Google Maps of the second run of track 2 using magnetometer-based heading tracker.	57
4.18	Representation in Google Maps of the second run of track 2 using gyroscope-based heading tracker.	58

List of Tables

2.1	Comparison between some of GPS receivers for bike usage.	20
4.1	Results of distance calculation with odometer.	45
4.2	Results of route calculation with gyroscope-based heading.	46
4.3	Results of route calculation with magnetometer-based heading.	48
4.4	Cuts made to the GPS signal in the first run of track 1, with magnetometer-based heading.	51
4.5	Cuts made to the GPS signal in the first run of track 1, with gyroscope-based heading.	53
4.6	Cuts made to the GPS signal in the second run of track 1, with magnetometer-based heading.	54
4.7	Cuts made to the GPS signal in the second run of track 1, with gyroscope-based heading.	54
4.8	Cuts made to the GPS signal in the first run of track 2, with magnetometer-based heading.	56
4.9	Cuts made to the GPS signal in the first run of track 2, with gyroscope-based heading.	56
4.10	Cuts made to the GPS signal in the second run of track 2, with magnetometer-based heading.	58
4.11	Cuts made to the GPS signal in the second run of track 2, with gyroscope-based heading.	58

LIST OF TABLES

Abbreviations

AGPS	Assisted GPS
BLE	Bluetooth Low Energy
EKF	Extended Kalman Filter
GLONASS	<i>Globalnaya Navigatsionnaya Sputnikovaya Sistema</i>
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
JS	JavaScript
INS	Inertial Navigation System
ROS	Robot Operating System
UKF	Unscented Kalman Filter

Chapter 1

Introduction

In this chapter, it is presented the problem which originates all of this dissertation: its context and motivation, the goals which are expected to achieve with this work and, at the end of the chapter, a brief description of how the document is structured.

1.1 Problem context and motivation

GPS is a wide-spread navigation system based on a satellite network that provides accurate location and time. Nowadays, it is becoming the norm for recreational and professional cyclists to log their rides' data using GPS receivers, and there are several services (e.g., Strava ¹, MapMyRide ², Garmin Connect ³) that store and allow the athletes to analyse their data.

Although GPS is the most popular navigation system used today, it requires an unobstructed line of sight to, at least, four satellites. Such is not always possible: high buildings (or being inside them), bridges, dense forests can weaken or totally block the GPS signal.

Even if it is not critical to have some signal loss in the case of data log for athletes (although it may be frustrating), there are some cases where it can be much more dramatic, such as in the Syndromic Surveillance project. This is a project being carried out by the African Community Health Workers, which consists in early detection or prediction of diseases in isolated populations on Sub-Saharan Africa (see appendix A). Currently, in Africa, millions of people suffer everyday from diseases, which propagate easily due to poor life and healthcare conditions.

¹See <https://www.strava.com/> for more information.

²See <http://www.mapmyride.com/> for more information.

³See <https://connect.garmin.com/> for more information.

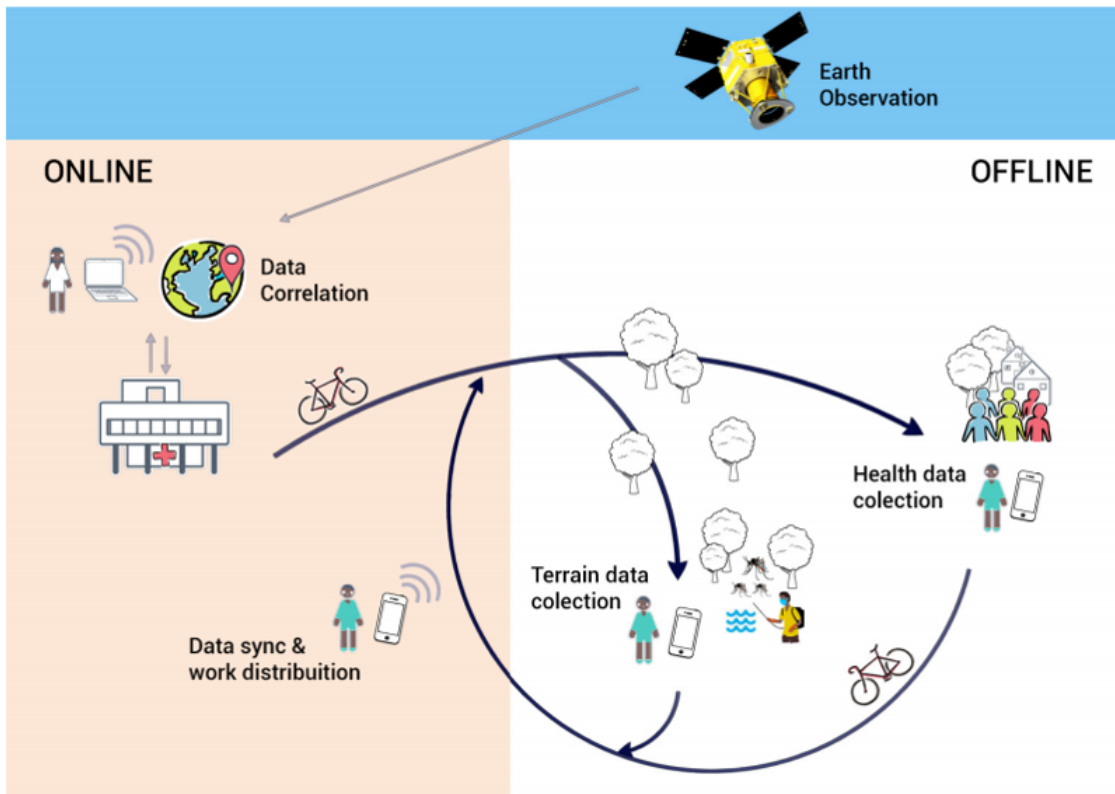


Figure 1.1: Diagram of the Syndromic Surveillance project.

Those volunteer workers move between isolated populations by bike, and need to log their data associated to certain coordinates, but GPS signal is not always available due to the existence of dense forest or mountainous regions and, when available, it takes much longer to "fix", due to the inexistence of 3G networks in the area to provide Assisted Global Positioning System (AGPS) data.

1.2 Goals

The idea is to create a cheap, lightweight solution for bikes that can use an inertial navigation system (INS), or, in other words, a system that uses motion and rotation sensors to estimate position, orientation and velocity of the cyclist. Nowadays, all smartphones have (at least) these sensors, and they are much cheaper and available than any other kind of sensors (e.g. LIDARs, SONARs).

To achieve this goal, the work in the course of this thesis consists in applying concepts of sensor fusion and signal processing to the data received from the smartphone's sensors to calculate the position, speed and orientation of the cyclist, and show all of this using a map service (e.g. Google Maps, OpenStreetMaps).

1.3 Document structure

The document starts by making a review of the already existing techniques and technologies for localization of objects or persons, including the sensors needed to achieve it. After that, based on the conclusions retrieved from that review, it describes the created approach to solve the problem introduced. After describing the requirements and system architecture, it introduces the created test framework and the experiments made with that same framework, discussing the obtained results. Finally, some final remarks are made about the project as a whole, and some ideas for future work are described.

Introduction

Chapter 2

Literature review

In this chapter, an overview over some sensors, techniques and algorithms related to the dissertation topic will be made, many of whose will be useful for achieving the goals established in the previous chapter.

2.1 Useful Sensors for Localization

Here, some of the sensors available for localization will be briefly presented and described, by going through how they work and some of their strengths and weaknesses.

2.1.1 Sensors available in smartphones

Nowadays, technological advances in integrated circuits have made possible a miniaturization of several devices, including sensors. This evolution opened new doors to generalization of cheap sensors, especially embedded on smartphones, which people carry everyday with them. Due to this generalization, and the fact that these sensors can be used for localization, a few notes about them will be presented here.

2.1.1.1 Gyroscope

Gyroscopes are devices which take advantage of the Earth's gravity to calculate the orientation of the device. They consist in a freely-rotating disk called rotor, mounted onto a spinning axis, standing on the centre of a larger and more stable wheel. This way, as the axis turns, the rotor remains stationary, indicating the central gravitational pull (see Figure [2.1](#)).

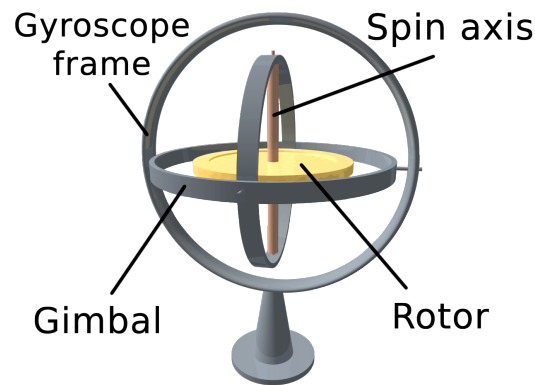


Figure 2.1: Representation of the gyroscope architecture.

Gyroscopes have a variety of applications, such as wireless computer pointing devices, allowing the user to control the mouse on air instead of using it on a flat surface; in robotics, where they are used to keep complex robots from falling, in virtual reality devices (such as Oculus Rift), to detect head movements; even in aircrafts, where they are used in the artificial horizon gauge, indicating to the pilot the position of the aircraft relative to the horizon.

2.1.1.2 Accelerometer

Accelerometers are devices which detect variations in the accelerations felt by the device - when in rest, a device is only under influence of one (constant) acceleration: g , the Earth's gravity, which points towards the centre of the planet. Since every movement causes a speed variation, it creates an acceleration (which is, as known the concept of speed variation over time), detected by this device. With that information, speed and displacement can be calculated by integrating the accelerations, respectively, one and two times.

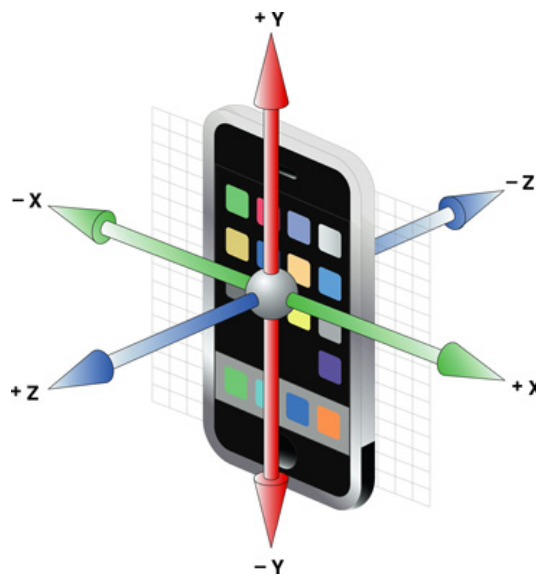


Figure 2.2: Representation of the accelerometer axis in smartphones.

Accelerometers, together with gyroscopes, are present in aircraft inertial navigation systems, which aim to improve the GPS signal quality, especially under bad reception conditions.

2.1.1.3 Magnetometer

As seen before, gyroscopes can be used to calculate the orientation of an object, but after a certain period of time, they accumulate too much error to be reliable. Another way to measure this orientation is through compasses, which take advantage of the Earth's magnetic field (see figure 2.3)

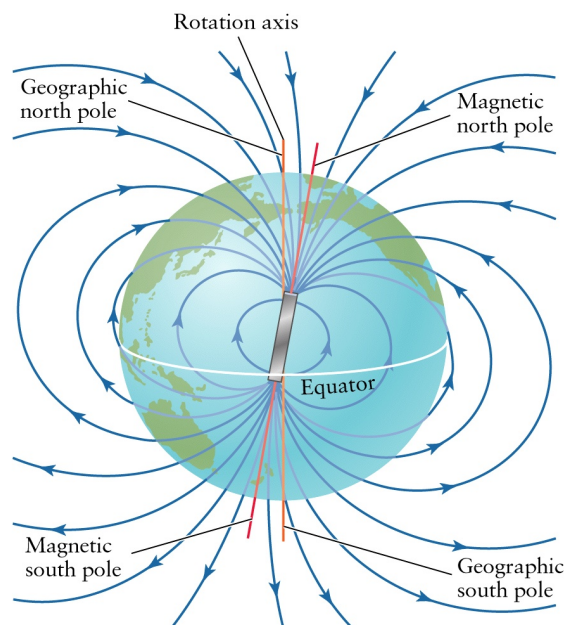


Figure 2.3: The Earth's magnetic field.

Although they are practically useless while indoors, since the noise caused by electronic appliances is too big, they are very accurate outdoors, as long as they are not used near metal surfaces or other strong magnetic fields, like high-tension electrical wires.

2.1.1.4 Barometer

Barometer is a sensor that measures the current atmospheric pressure. Originally developed by Evangelista Torricelli, the idea is to use an incompressible liquid (such as mercury) in a dish, and place an inverted measuring cylinder on top of it (as shown on figure 2.4).

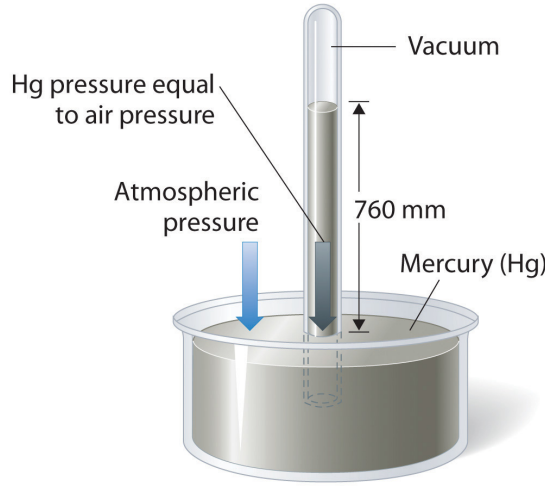


Figure 2.4: Example of mercurial barometer montage.

The force exerted by the atmosphere on the liquid forces it to rise inside the cylinder. By measuring this height, the air pressure can be calculated through the Fundamental Law of Hydrostatics (equation 2.1)

$$p - p_0 = \rho gh \quad (2.1)$$

in which p and p_0 , are respectively the pressure measured at the two reference points (bottom and top of the column, in this case), ρ is the volumetric mass density of the liquid, g is Earth's gravity acceleration, and h is the height of liquid inside the column. Since the pressure above the mercurial is negligible when compared to the atmospheric pressure, the expression can be simplified to

$$p_{atm} = \rho gh \quad (2.2)$$

where p_{atm} is the atmospheric pressure.

Pressure Altimeter A barometer can be used to calculate the current altitude. Since the pressure exerted by the atmosphere becomes smaller as the altitude rises (the air becomes "thinner", i.e., there are less air particles), these two values can be related through the expression 2.3

$$h = \left(\frac{RT}{gM} \right) \cdot \ln \left(\frac{p_0}{p} \right) \quad (2.3)$$

where p_0 is the pressure at the starting height, p is the pressure at the measurement height, h is the altitude, g is the Earth's gravity acceleration, M is the molecular weight of the fluid (in our case, the apparent molecular weight of the air), R is the gas constant, and T is the temperature of the air (in Kelvin). This line of thought, however, makes some assumptions that contribute to the accumulated error of the altimeter, such as considering the air composition and temperature

constant. To prevent drifts in the order of dozens or even hundreds of meters, some correction techniques must be employed for each one of these parameters. ([Jackson & Crocker, n.d.](#))

Barometers in smartphones High-end smartphones such as Samsung Galaxy S5, Motorola Nexus 6 or Apple iPhone 6 have built-in barometers, mainly for aiding GPS to have a quicker time to fix, by providing an estimate to the receiver's height. These do not necessarily rely on the same technique described previously to calculate pressure; other techniques such as using the piezoresistive effect (which detects a change in the electrical resistivity of a semiconductor or metal when a mechanical strain is exerted to them) can be applied.

These kind of sensors have been applied successfully to detecting considerable variations in height, such as the ones that occur when a person moves from one building floor to another. However, it is important to note that, due to the error inherent to the sensor, the height between floors must be relatively elevated - at least 1.7 meters. ([Muralidharan, Khan, & Misra, 2014](#))

MIT researchers have also achieved positive results in using the barometer as a low-power technological alternative to detect whether a person is idle, walking, or moving on a vehicle by detecting patterns in height variations. ([Sankaran et al., 2014](#))

2.1.1.5 Camera (Visual odometry)

By using a camera attached to a vehicle, capturing images at a certain rate and using some post-processing techniques, optical flow can be detected and used to estimate the speed (and consequently, the distance) during that time period. This can be achieved through computer vision techniques, using libraries like OpenCV. Several methods can be used, such as the Lucas-Kanade method, the Horn-Schunck method, the Buxton-Buxton method or Black-Jepson method.

All of the mentioned methods before use colour brightness to determine optical flow. But this information alone is not enough to calculate this flow, so they all make some assumptions about the image itself, e.g, the Lucas-Kanade method assumes that the displacement of the contents between two nearby frames is small and approximately constant within the neighbourhood of the considered point.

In order to bypass this kind of assumptions on the image contents, many researchers have been trying to use not only gray information, but also color one. ([Xiang, Peng, & Zhang, 2009](#)) ([Aires, Santana, & Medeiros, 2008](#))

Advances in optical flow techniques are particularly interesting for video compressing algorithms, which try to detect sequences of similar frames to only store one of them, reducing the amount of space needed. Since the OpenCV library has a poor performance for detecting optical flow on mobile phones, researchers have successfully developed a system for smartphones that uses this kind of visual odometry, by reusing the dedicated hardware for video compression embedded on the phone, used to encode the video stream captured by the camera. ([Bitsch Link, Gerdsmeyer, Smith, & Wehrle, 2012](#))

2.1.2 Sensors not available in smartphones

2.1.2.1 Odometer

The fundamental idea behind odometry is incrementally collecting and integrating motion information about the travelling object, thus giving accurate short-term measures at a high sampling rate with a very low cost. These factors make this system a widely used navigation method, especially for small robotic vehicles.

Odometry in bikes For bikes, wheel-based odometers are available. There are several brands and models in the market, but all of them rely on the same principles. They have at least two parts: a sensor, mounted in a wheel (typically the front wheel), and a computer, which processes the signal coming from the sensor and outputs the information to the cyclist.



Figure 2.5: An example of an odometer for bikes. On top, the computer, at the bottom, the magnets that capture the signals for speed and distance calculations.

These two parts, acting together, can determine accurately the bicycle "instantaneous" speed (the sampling rate is high enough to assume that). The sensor, after each wheel revolution, sends a signal to the computer (through a wired or wireless connection). The computer, by its turn, through measuring the time interval between signals, can calculate the angular speed of the wheel. By knowing this information and the wheel radius (configured by user before usage), it can be converted to linear speed, as shown in equations 2.4 and 2.5. Using the wheel radius, its perimeter can also be calculated, which may be used to compute the travelled distance.

$$\omega = \frac{2\pi}{T} \quad (2.4)$$

$$v = \omega r \quad (2.5)$$

2.1.2.2 LiDAR

LiDAR refers to a remote sensing technology used to compute distances to objects by emitting intense and focused beams of laser light and measuring the time until their reflection. The three-dimensional coordinates of the object are then computed using the time interval between the light pulse emitted and the reflection received, the angle at which the pulse was emitted and the absolute location of the sensor. The latter forces the object containing the LiDAR to have either a GNSS receiver (such as GPS) and/or an inertial navigation unit, in order to accurately determine the current sensor position.

High-end LiDARs are very accurate devices, being used in accuracy-critical systems and tasks, such as mapping the surface of the Earth and object detection in Google's self-driving car (see Figure 2.6). Despite this elevated accuracy, these kind of sensors have a very high price, not affordable to everyone.



Figure 2.6: Google's self-driving car. On top of it, the LiDAR is visible - it continuously rotates 360 degrees to detect obstacles in its surrounding environment.

2.1.3 Sensor fusion

Sensor fusion is the set of techniques which allow data provided from several different sensors about the same measures to be combined in order to create one single value for each of the measures. Sensor fusion techniques vary in complexity and capabilities. In this section, two topics will be explained: the first, more concrete to the problem, relies on fusing gyroscope and accelerometer data to project this data into the global frame, i.e. the Earth axis; the second, more generic, is about the filters used for several kinds of sensors and models. However, only two algorithms will be reviewed: the Kalman Filter and the Extended Kalman filter, since they are widely used in Inertial Navigation Systems (INS).

2.1.3.1 Projecting accelerometer and gyroscope into global frame

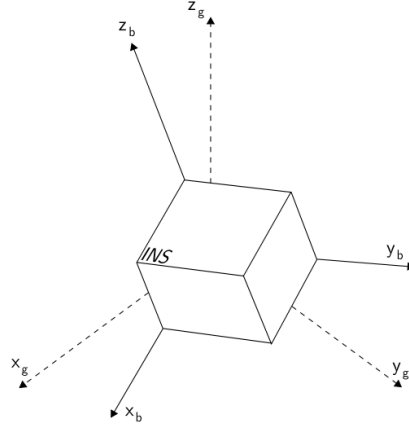


Figure 2.7: Representation of the body and global frame.

In order to use the information produced by the accelerometer and gyroscope, one must project the obtained measurements into the Earth's global axis. For that, first, the integration of the angular speeds provided by the gyroscope must be projected to the global frame and integrated in order to use as reference for projection and integration of accelerometer data.

Gyroscopes The output of gyroscopes, a three-dimensional angular velocity signal (also known as attitude) $\omega_b(t) = (\omega_{bx}(t), \omega_{by}(t), \omega_{bz}(t))^T$, with ω_b values being the angular velocity for each axis in the body frame can be used, along with the direction cosines representation from this attitude relative to the global frame (see Figure 2.7), to specify a 3x3 rotation matrix $C(t)$, defined through equations 2.6 and 2.7. (Woodman, 2007)

$$\Omega(t) = \begin{pmatrix} 0 & -\omega_{bz}(t) & \omega_{by}(t) \\ \omega_{bz}(t) & 0 & -\omega_{bx}(t) \\ -\omega_{by}(t) & \omega_{bx}(t) & 0 \end{pmatrix} \quad (2.6)$$

$$C(t) = C(0) \cdot \exp\left(\int_0^t \Omega(t) dt\right) \quad (2.7)$$

The only problem here is that the signal coming from the gyroscopes is discrete (usually with a static sample rate) instead of being continuous, which disallows the usage of the expression presented above without any adaptations. One of the techniques which can be used is the rectangular rule, a low order scheme to calculate the definite integral of a function through approximation in rectangles (calculating the area of each rectangle and sum all of them), which is sufficient for low accuracy applications. By using this rule, the interval $t + \delta t$, and $\omega_b = (\omega_{bx}, \omega_{by}, \omega_{bz})^T$ can be considered as the sample. Having

$$B = \begin{pmatrix} 0 & -\omega_{bz}\delta t & \omega_{by}\delta t \\ \omega_{bz}\delta t & 0 & -\omega_{bx}\delta t \\ -\omega_{by}\delta t & \omega_{bx}\delta t & 0 \end{pmatrix} \quad (2.8)$$

and $\sigma = |\omega_b \delta t|$, the following equation can be written: (Woodman, 2007)

$$C(t + \delta(t)) = C(t) \left(I + \frac{\sin(\sigma)}{\sigma} \cdot B + \frac{1 - \cos(\sigma)}{\sigma^2} \cdot B^2 \right) \quad (2.9)$$

which is the update equation of the rotation matrix each time new data is available.

Similarly to gyroscopes, the output coming from accelerometers $a_b(t) = (a_{bx}(t), a_{by}(t), a_{bz}(t))^T$ can be projected to the global frame, by using the rotation matrix $C(t)$ coming from gyroscopes (Woodman, 2007):

$$a_g(t) = C(t)a_b(t) \quad (2.10)$$

By removing the gravity acceleration, the remaining one can be integrated to calculate velocity, and once again to obtain displacement:

$$v_g(t) = v_g(0) + \int_0^t a_g(t) - g_g(t) dt \quad (2.11)$$

$$s_g(t) = s_g(0) + \int_0^t v_g(t) dt \quad (2.12)$$

As with gyroscopes, accelerometers do not provide a continuous measure, only discrete ones. So, an integration scheme must be used, like before, to use this principle. Again, using the rectangular rule, the following equations can be written (Woodman, 2007):

$$v_g(t + \delta t) = v_g(t) + \delta t \cdot (a_g(t + \delta t) - g_g) \quad (2.13)$$

$$s_g(t + \delta t) = s_g(t) + \delta t \cdot v_g(t + \delta t) \quad (2.14)$$

2.1.3.2 Kalman filter

The Kalman filter was originally developed by Rudolf Emil Kálmán, which, in 1960, presented a solution to the discrete data filtering problem. This technique combines all the available measurements from the system with some previous knowledge about that same system. (Merwe, 2004)

Modelling the Process With this combination of indirect, inaccurate and uncertain observations, it can infer an optimal estimate of the current state variables. However, for this estimate to be optimal, the system is required to be linear, and the noise originating from the measurements (and

process) to be white and Gaussian (usually distributed with mean zero and standard deviation *sigma*), i.e, noise has infinite energy (non-real approach, but simplifies calculations), and that its values are not correlated with time, meaning that knowing the noise value at a certain time instant does imply the remaining values can be estimated.

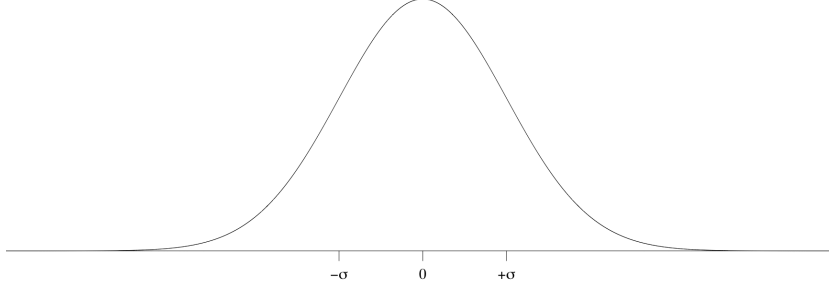


Figure 2.8: Gaussian (or normal) distribution curve.

Based on these assumptions, discrete-time process can be modelled using the following equations:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \quad (2.15)$$

$$z_k = Hx_k + v_k \quad (2.16)$$

where $x_k \in \mathbb{R}^n$, with x containing the n variables which must be computed; A is the $n \times n$ matrix which exists between the previous x_{k-1} state and the current one, x_k (this must be a linear relationship); B is the $n \times l$ matrix which relates the system's entries, $u_k \in \mathbb{R}^l$, with the state x ; H is the $m \times n$ matrix which relates x with the measures $z_k \in \mathbb{R}^m$. Since the Kalman filter assumes a stochastic approach of the process to be estimated, two more variables, w_k and v_k appear in the previous equations, which, respectively, model the error resulting from process and measurements. Their covariance matrices are as follows:

$$E[w_k w_i^T] = \begin{cases} Q_k, & i = k \\ 0, & i \neq k \end{cases} \quad (2.17)$$

$$E[v_k v_i^T] = \begin{cases} R_k, & i = k \\ 0, & i \neq k \end{cases} \quad (2.18)$$

$$E[w_k v_i^T] = 0, \forall k, i \quad (2.19)$$

Algorithm The filtering process has two main stages: prediction and correction. In the prediction stage, the filter projects in the following instant the current state (\hat{x}_{k-1}) and the error covariance (P_{k-1}) in order to produce an *a priori* estimate for the following instant (\hat{x}_k^- and \hat{P}_k^-), given

by equations 2.20 and 2.21.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \quad (2.20)$$

$$\hat{P}_k^- = AP_{k-1}A^T + Q \quad (2.21)$$

The correction stage starts by calculating the Kalman gain, an $n \times m$ matrix (K), which aims to minimize the *a posteriori* error covariance, stated in equation 2.22. Next, the algorithm integrates the measurements in the *a priori* calculated state estimate through equations 2.23 and 2.24, obtaining an *a posteriori* estimate for state variables and error covariance value. (Xavier, 2011)

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \quad (2.22)$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \quad (2.23)$$

$$P_k = (I - K_k H)P_k^{-1} \quad (2.24)$$

Initialization and Tuning In order to use the Kalman filter, the R and Q matrices must be previously defined and variables \hat{x}_{k-1} , P_{k-1} must be initialized. The R matrix, representing the covariance of the observation (through measurements) noise is easy to obtain, by collecting noise samples and using them to calculate its error covariance before stating to use the filter. The Q matrix, which represents the covariance of the process noise, is trickier to calculate, since it is not possible to directly observe the process which will be estimated. A possible approach, also extendible to matrix Q , relies on guessing its values and adjusting them according to the calculated error.

Lastly, \hat{x}_{k-1} and P_{k-1} ; \hat{x}_{k-1} initial guess must be as close to the expected value as possible. The initial error covariance value will define how the state estimate converges to the real value. (Xavier, 2011)

2.1.3.3 Extended Kalman filter

Since not all processes can be defined by linear systems, as assumed by the Kalman filter, an alternative approach can be used: the Extended Kalman filter (EKF). It linearises the state around the current mean and covariance using partial derivatives of the process and observations functions. (Xavier, 2011)

Process modeling As stated before, the EKF applies to process which cannot be defined by linear systems. Given this, our models for state and measurements are no longer linear approximations, but non-linear functions as described by equations 2.25 and 2.26, where f and h are non-linear functions. (Xavier, 2011)

$$x_k = f(x_{k-1}, u_k, w_{k-1}) \quad (2.25)$$

$$z_k = h(x_k, v_k) \quad (2.26)$$

Algorithm Based on the previous assumptions, it can be demonstrated that the prevision equations of EKF are as follows:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_k, 0) \quad (2.27)$$

$$\hat{P}_k^- = A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \quad (2.28)$$

where A is the Jacobian Matrix of partial derivatives of f with respect to x and W is the Jacobian matrix of partial derivatives of f with respect to w . (Xavier, 2011)

The correction equations of the EKF can be described as written below:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \quad (2.29)$$

$$\hat{k}_k = \hat{k}_k^- + K_k (z_k - h(\hat{x}_k^-, 0)) \quad (2.30)$$

$$P_k = (I - K_k H_k) P_k^- \quad (2.31)$$

where V is the Jacobian matrix of partial derivatives of h with respect to v . (Xavier, 2011)

2.1.3.4 Multiple Kalman filters

Kalman filters are popular for sensor fusion due to their simple formulation, which makes them easy to adapt for that purpose. Despite this simplicity, they provide significant improvements on position accuracy. (Abreu, Xavier, Castro Silva, Reis, & Petry, 2014)

However, the Kalman filter only works well within a certain framework and under certain conditions; for instance, if the system architecture is modular and needs to have the ability to add or remove sensors without modifying its algorithms, a single Kalman filter, as-is, cannot provide that level of flexibility; also, different sensors may provide data at different rates, and by collecting data based on the rate of a single sensor is not feasible, since much of the remaining information will be lost. To overcome these difficulties, an architecture of multiple Kalman filters was imagined by a group of researchers, which consists in using multiple Kalman filters, each one using a combination of different sensors. The number of filters is determined by several parameters, such as the number of dimensions represented by each filter, or the transformations which are needed to apply to the data before sending it to the filter.

Using this technique brings another advantage: dealing with redundant information. For example, considering a system which tries to calculate its position by fusing data from an accelerometer with data coming from two different positioning systems, two redundant measures of absolute position will be provided. This can be solved by using two identical Kalman filters, each one combining the accelerometer with a different position system. Then, in order to integrate the two estimates, its error covariance can be used to calculate a final estimate using a weighted arithmetic mean, as described on equation 2.32:

$$X_g = \frac{\frac{X_1}{P_1} + \frac{X_2}{P_2}}{\frac{1}{P_1} + \frac{1}{P_2}} \quad (2.32)$$

where X_g is the global fused estimate, X_n are the estimates provided by each filter, and P_n are their respective covariance matrices. This equation ensures that, the smaller the error covariance of an estimate, the bigger its contribution to the global estimate. (Drolet, Michaud, & Cote, 2000)

2.2 Positioning techniques

In order to use the information provided by the sensors shown before, it is important to analyse some algorithms that make use of such information to provide estimates of the user's current position.

2.2.1 Absolute positioning techniques

Absolute positioning techniques are the ones which determine the location of a place with respect to certain coordinates which themselves have a fixed reference (such as the Earth's latitude and longitude system).

2.2.1.1 Active Beacons

The active beacons positioning technique relies on an infrastructure of external signal transmitters - the beacons - which are mounted on well-known positions. A receiver who detects signals from, at least three of these beacons can then accurately calculate its position. This approach, although being widely used in airplanes and boats for its reliability and accuracy (using minimal processing), has elevated costs for building and maintaining all the beacon infrastructure.

Two different types of active beacons systems exist: trilateration and triangulation.

Trilateration Trilateration uses distance measurements from the receiver to each one of the three beacons to calculate the receiver's position. By intersecting the three circumferences centred on each of the detected beacons, with radius equal to the measured distance, a single point results from there - the exact position of the receiver (see Figure 2.9). (Cook, Buckberry, Scowcroft, Mitchell, & Allen, 2005)

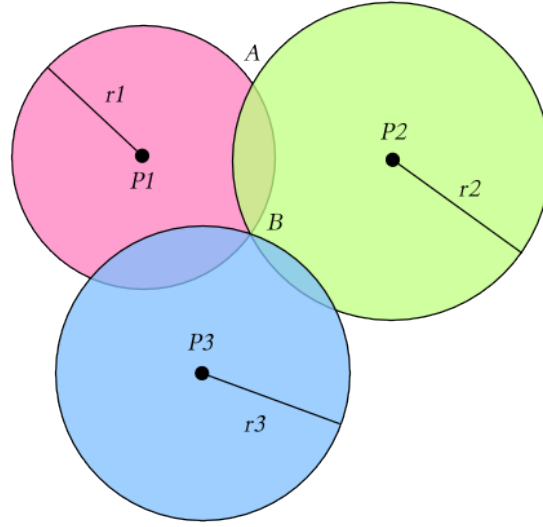


Figure 2.9: Trilateration example.

Triangulation Instead of using absolute distance to each of the beacons, triangulation measures angles between a reference direction, using, for example, a rotating receiver. This alone is enough to calculate the coordinates of the receiver position (see Figure 2.10). (Pierlot, Urbin-Choffray, & Van Droogenbroeck, 2011)

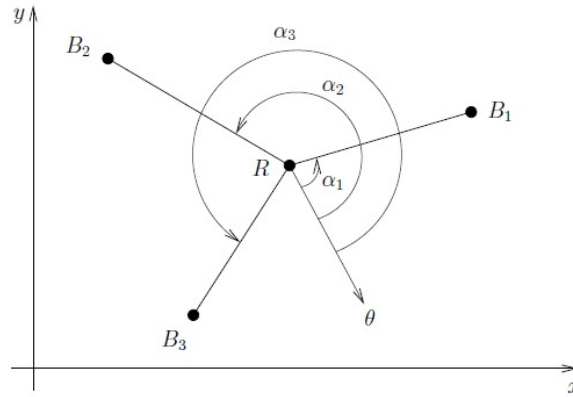


Figure 2.10: Triangulation example using three beacons. In the figure, R is the receiver, B_1, B_2 and B_3 are the beacons, and, respectively α_1, α_2 and α_3 are the angles measured from the rotating receiver R to each one of the beacons.

2.2.2 Global Navigation Satellite Systems

GNSSs are an adaptation of trilateration systems. Using satellites placed in orbit around the Earth, the distance to each of the satellites is then measured using time differences, since each satellite sends a periodic signal containing its time of transmission, which the receiver then can compare to its own time and, based on that time interval, calculate the distance to that satellite. For calculating its position, a receiver must have a clear, unobstructed view of, at least, four satellites: 3 of them

to calculate latitude, longitude and altitude, and the extra one to measure time drift (important in error calculation - if the drift is too high, the calculations will not be valid - in that case, there is not a "fix"). By intersecting the spheres formed by the other three satellites, two points are calculated. Since one of them may always be discarded for being outside the Earth's surface, the point remaining the receiver's location. (Murphy, 2000, p. 208-209)

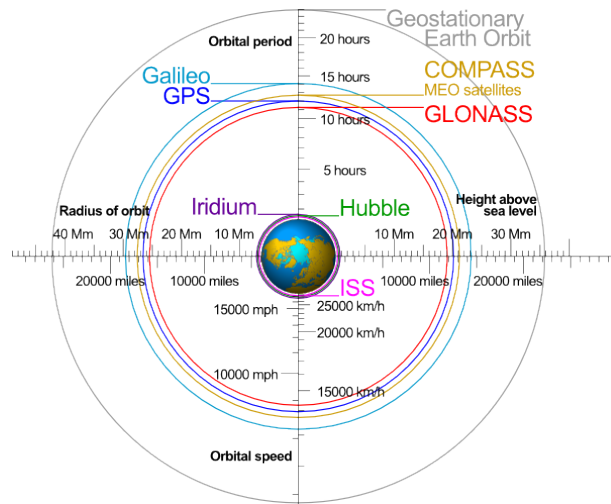


Figure 2.11: Orbits from each of the GNSSs' satellites.

GPS GPS is a GNSS initially created by the Department of Defence of United States of America in 1973. Being currently the most widespread used GNSS, it has a constellation of 27 operating satellites which orbit at a height of 20180km, with orbit periods of 11h58min. Although the USA government introduces intentional small drifts to the times in the satellites' atomic watches, to prevent its usage for high-precision guidance of enemy missiles, its mean precision is set to approximately 10-15 meters around the receiver. By cancelling that drift, using, for example, differential GPS, centimetre precision can be achieved. (Murphy, 2000; Bidikar, Rao, Ganesh, & Kumar, 2014; Facts, 2000)

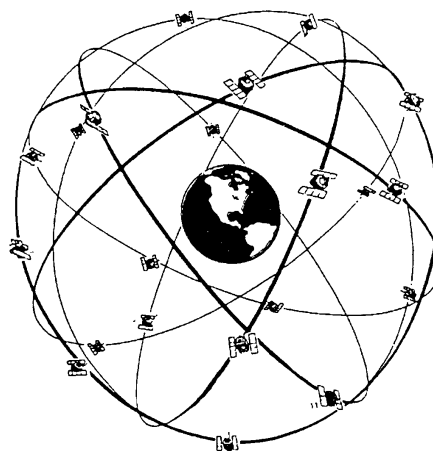


Figure 2.12: NAVSTAR GPS satellite network representation.

GLONASS In 1976, the former Soviet Union started the development of a concurrent GNSS to GPS - the GLONASS. Currently, GLONASS is operated by the Russian Federation. It has a constellation of 31 satellites, although only 24 of them are currently operational. These satellites orbit at a height of 19130km, with orbit periods of 12h38min. ([Facts, 2000](#))

Galileo Galileo is a GNSS project approved in 2010 by the European Union, which aims to provide an alternative high-precision positioning system for the EU citizens. At its full capacity, it will have a constellation of 30 satellites: 27 operational plus 3 active spares. They will orbit at a height of 23222km with orbit periods of 14h05min. ("[Galileo Fact Sheet](#)", 2013)

GNSS usage in bicycles Several products for specific usage in bikes are available. Amongst the more popular are Garmin (especially the Edge series) and Polar GPS receivers. Using GPS (and, in some cases, GLONASS) signal, all these devices can show the current speed and distance travelled, as well the elapsed time in the activity. Some of them, using embedded maps, can even show their current position and give turn-by-turn directions throughout a pre-established route (as show in Figure 2.13) Table 2.1 established a synthetic comparison between the most popular GPS receivers currently on the market.

Table 2.1: Comparison between some of GPS receivers for bike usage.

Brand	Garmin						Polar
Model	Edge 200	Edge 500	Edge 510	Edge 800	Edge 810	Edge 1000	V650
Launch date	Sept 2011	Dec 2009	Jan 2013	Nov 2010	Jan 2013	May 2014	Nov 2014
Data transfer	USB	USB	USB Bluetooth	USB	USB Bluetooth	USB Bluetooth Wi-Fi	USB
GNSSs	GPS	GPS	GPS GLONASS	GPS	GPS	GPS GLONASS	GPS
Battery life	14 hours	18 hours	20 hours	15 hours	17 hours	15 hours	10 hours
Altimeter type	GPS	Barometric	Barometric	Barometric	Barometric	Barometric	Barometric
Compass type	N/A	GPS	GPS	GPS	GPS	GPS	GPS



Figure 2.13: Garmin Edge 810 GPS receiver.



Figure 2.14: Polar V650 GPS receiver.

2.2.3 Map Matching

Map Matching is a technique where a representation of the environment is pre-built before the actual usage. Given a set of measurements done by the available sensors, that information is then compared to the one existing on the pre-existing map, thus calculating an estimate of the object's current location.

In robotics, where this technique is frequently used, the robot explores its local environment and builds this map using data gathered from its sensors for future usage. RViz, a tool for usage with ROS-compliant robots, is highly useful for this purpose. (Zaman, Slany, & Steinbauer, 2011)

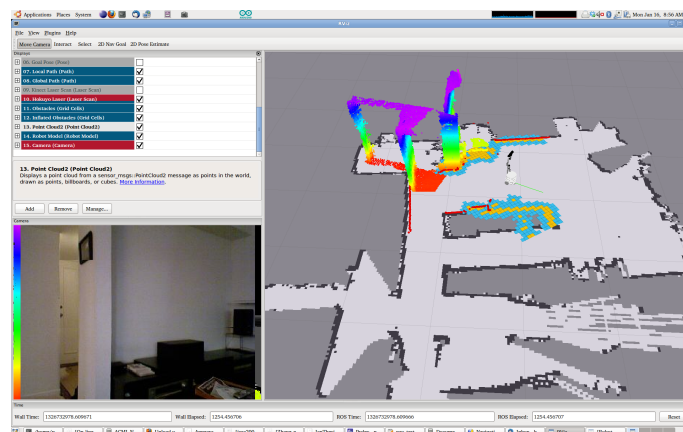


Figure 2.15: Map building using RViz.

Other kinds of approaches are the ones where these maps were built manually (using technologies like OpenStreetMaps). Together with all the terrain and buildings, several paths were created.

With this information, both algorithms could then estimate and predict, based on the position calculated through other sensors, the path the user was following. (Constandache, Choudhury, & Rhee, 2010; Link, Smith, Viol, & Wehrle, 2011)

2.2.4 Relative positioning techniques

Relative position techniques are the ones which compute location estimations in reference to certain landmarks or known locations.

2.2.5 Inertial Navigation

An inertial navigation system consists in using motion sensors - accelerometers and gyroscopes to measure the rate of acceleration and rotation of the associated object. Such values are then "integrated" once, for accelerometer data, or twice for gyroscope data, in order to calculate position, as shown in Figure 2.16.

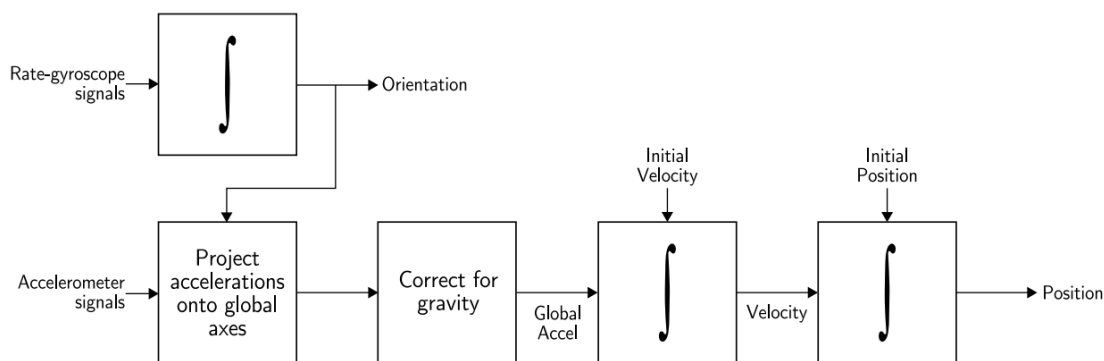


Figure 2.16: An example of an inertial navigation system architecture.

Accumulated errors Accelerometers and gyroscopes are widely available at low-cost prices, in our smartphones. However, despite being cheap, they are very noisy. That noise originates drifts from the real values, being then amplified by the numerical integration techniques that must be employed in order to obtain velocity and displacement. Figures 2.17 and 2.18 represent the calculated trajectory of a book tossed into the air with a INS localization system, and intend to demonstrate the drift between real positions and calculated ones. (*Integration Drift*, n.d.)

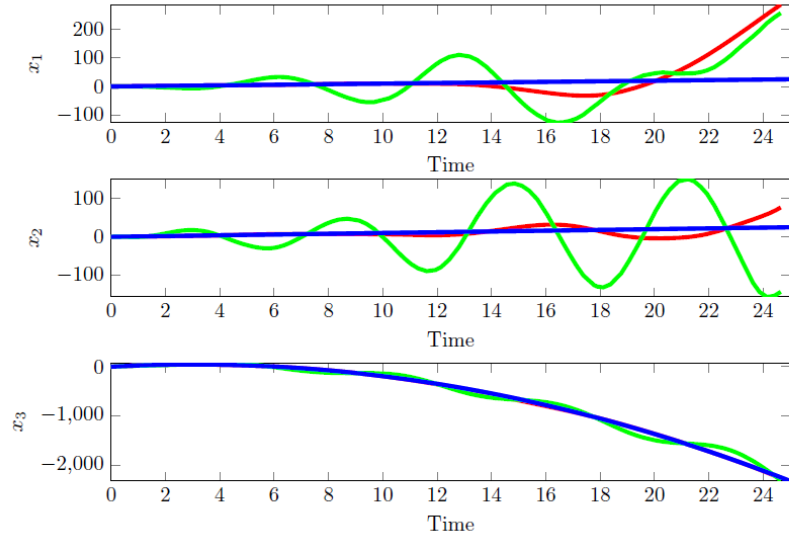


Figure 2.17: Actual positions (blue), calculated positions (red), calculated positions under noise (green).

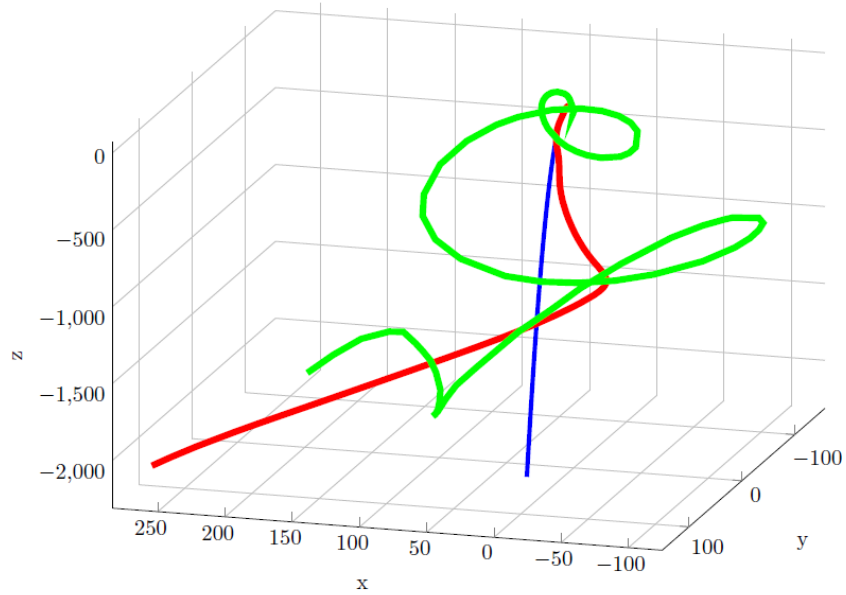


Figure 2.18: Actual trajectory (blue), calculated trajectory (red), calculated trajectory under noise (green).

To minimize this kind of problem, inertial navigation systems are often combined with other kinds of systems, such as Global Positioning Systems or Map Matching. (Gade, 2009) CompAcc and FootPath are two examples of indoor location systems built with this kind of combination. (Constandache et al., 2010; Link et al., 2011)

2.3 Earth distances

To be able to compute positions and make conversions between displacements in rectangular coordinates and Earth coordinates (latitude, longitude, altitude), it is important to analyse some of the abstractions made to represent the Earth, and how this affects the error for each one of them.

2.3.1 Abstractions and representations

Earth, like other planets, has a round surface (although it is not a topographically smooth one). Despite being sufficient for small distance calculations, the flat model of the Earth is inadequate for longer measurements and other kinds of usages, like mapping the planet's surface, since calculations drift too much from real distances. The spherical Earth model, proposed by Pythagoras, provides a surface which simplifies many calculations and is satisfactory for many purposes. However, for very long measurements (e.g, measuring continents and oceans), an ellipsoid or geoid representation are more accurate.

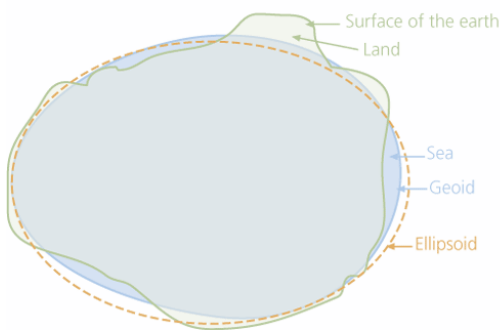


Figure 2.19: Geoid and ellipsoid representation models.

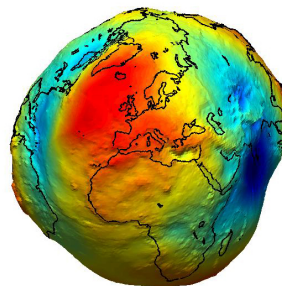


Figure 2.20: Geoid representation example. The geoid represents the Earth's gravitational potential. This potential is not uniform, depending on the material type and distribution underneath the surface. In this image, the warmer the colour, the higher the potential.

However, choosing a 3D model of the Earth is only half of the problem. In order to represent it visually on our 2D screens and paper sheets, an adequate projection of it must be made. There are several proposed solutions, based on several properties, such as area, shape, direction, bearing, distance or scale. Typically, map projections conserve at least one of these properties. Some commonly used map projections are Stereographic, Lambert Conformal Conic, Mercator (2.21), Robinson and Transverse Mercator. (Snyder, 1987)

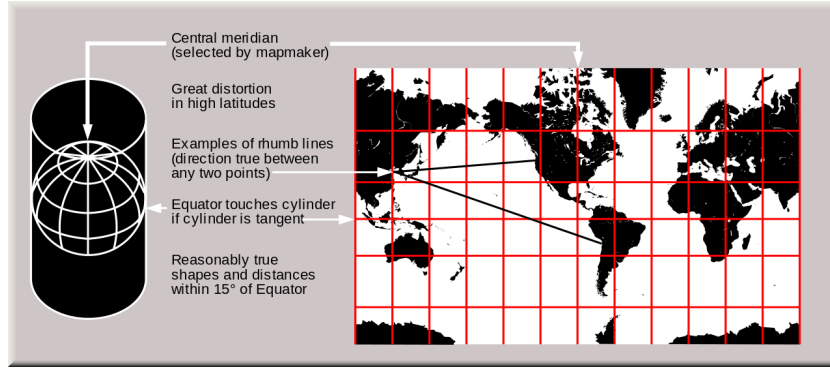


Figure 2.21: Mercator projection of the Earth surface.

2.3.2 Haversine formula

Using a non-flat model to represent the surface of the Earth disallow conversion of coordinates to rectangular ones and use the Euclidean distance equation described in 2.33 order to calculate the distance between two points $p(p_1, p_2, p_3)$ and $q(q_1, q_2, q_3)$. (Vincenty, 1975)

$$d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + (q_3 - p_3)^2} \quad (2.33)$$

Several approaches to this problem are formulated, with different degrees of precision (and intended for usage if different distance ranges). Among them is the Haversine formula (based on the law of haversines), described by equation 2.34, where ϕ_1 is latitude of point 1, ϕ_2 is latitude of point 2, λ_1 is longitude of point 1, λ_2 is longitude of point 2 and r is the radius of the Earth.

$$d = 2r \arcsin \left(\sqrt{\sin^2 \left(\frac{\phi_2 - \phi_1}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right) \quad (2.34)$$

Although it assumes a spherical Earth, it provides a near-perfect distance calculation for relatively small distances. (Ivis, 2006)

Literature review

Chapter 3

Solution Architecture

After identifying and defining the problem boundaries, and having researched about the state of the art in the domain, in this chapter, each component of the created solution will be presented.

3.1 Solution Requirements

In order to solve the problem presented in chapter 1, some requirements needed to be established in such a way that would make the implementation feasible during the dissertation time. In a functional level, a system with capability to estimate near real-time positions without GPS signal, in an area of 1000 m^2 around the true position was needed. In a more technical level, such system should reuse at most the sensors already available in smartphones and should reuse, as possible, internal components and code structure already implemented by Fraunhofer to deal with this kind of sensors; and the system should be compatible to use with Android 4.3+.

3.2 Architecture Overview

Given the objectives previously defined and the sensors analysed in chapter 2, a solution was developed integrating:

- an Inertial Navigation System - since smartphones are largely used worldwide (although, as expected, more in first-world countries), the hardware already embedded in these devices can be reused, namely its accelerometer, gyroscope, magnetometer and barometer. Gyroscope and magnetometer are used to calculate heading. Barometer (when available) is used to calculate altitude through air pressure, since it is more accurate than detecting altitude variations through the accelerometer/gyroscope pair. Other alternatives, such as active beacons would require an expensive structure of beacons which do not comply with the defined requirements. Map matching would be interesting to use, through usage a mapping platform such as Google Maps or OpenStreetMaps underneath: unfortunately, connection to

Solution Architecture

the Internet is not always guaranteed, and in Sub-Saharan Africa, there are no roads from which create well-defined waypoints to use mathematical optimizations like the ones used, for example, by Strava¹. Using Visual Odometry would require high levels of processing capabilities, which would not be in reach of every smartphone.

- an odometer attached to the bicycle being monitored, connected to the smartphone via Bluetooth Low Energy (BLE) which is processing and integrating its data. As previously explained in chapter 2, the phone sensors suffer from high noise, which directly affects the calculation of displacement (see figure 2.17); to have better accuracy, the odometer outputs more precise readings of distance and instant speed, improving the overall system precision and effectiveness. Besides BLE sensors, there are also ANT+² sensors available, which, due to the protocol used, have advantages and disadvantages over BLE sensors. However, its support for Android is very limited - the Android SDK does not have any native method to use ANT+, and a very limited number of high-end phones have support for it.

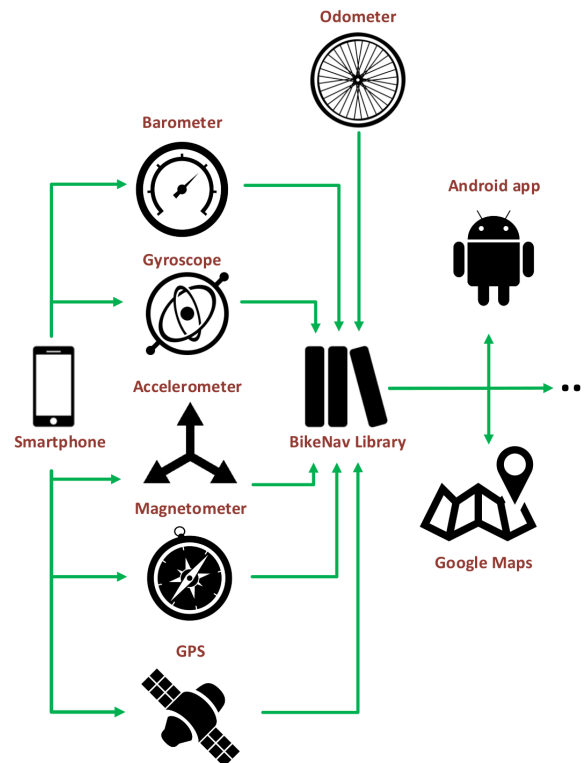


Figure 3.1: High-level overview of the solution architecture.

- a GNSS system (GPS), using the GPS receiver which is also available on smartphones; although it may not be available everywhere, due to satellite visibility issues, the system supports it, since its precision is high without the need for any additional components (the INS is, in short, a fallback mechanism when GPS signal is not available). Although systems

¹Slide to GPS data: <http://labs.strava.com/slide/>

²More information available at: <http://www.thisisant.com/consumer/ant-101/what-is-ant>

like GLONASS and GALILEO are being developed, GPS still remains the most reliable and most supported GNSS system available.

A library (BikeNav Library) which accepts data from these three sources of information was developed, acting as a position provider: it combines all these informations and, for each instant, computes the bicycle's current position. This component can then have many different kinds of usage: currently, in order to test the developed algorithms, it is subject to test runs created over unit testing platforms, which output the test results to locally forged web pages using Google Maps JavaScript (JS) API. However, the BikeNav Library can also be used as part of any other application which needs this kind of system.

3.3 BikeNav Library

This is the core component of the whole developed solution. It is a generic and reusable library, which works both on and outside of Android, since it was written in *vanilla* Java (i.e, without using methods or structures provided by the Android SDK). To achieve a higher degree of code extensibility flexibility and organization, three design patterns were employed: the observer pattern, the singleton pattern, and the factory pattern.

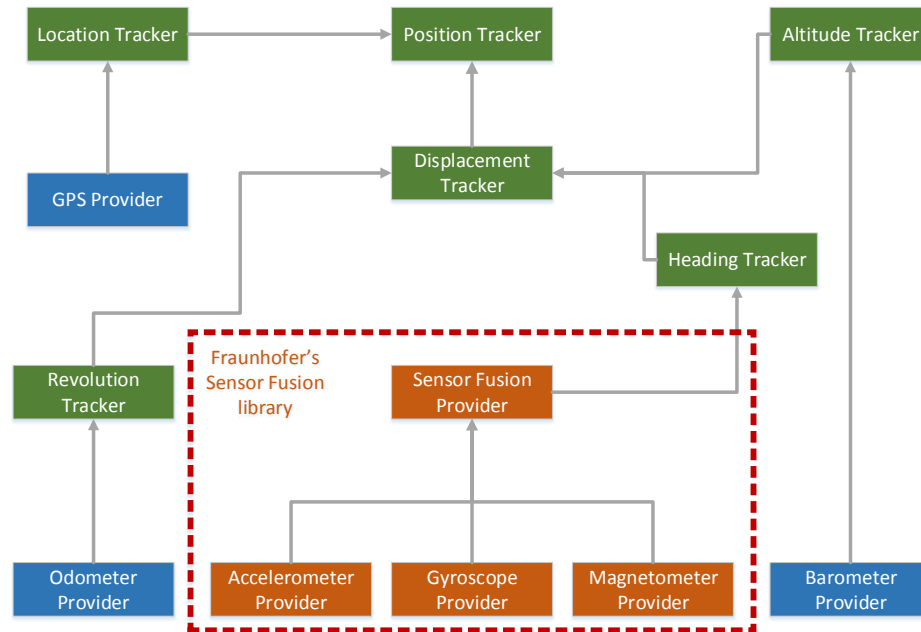


Figure 3.2: Overview of the BikeNav library system architecture. Sensor data providers are marked in blue (except for the ones already implemented by the Fraunhofer's sensor fusion library, which are marked in red). Trackers of relevant movement characteristics are marked in green.

The code is divided into several independent and easily replaceable modules: providers (presented in 3.3.1) or trackers (presented in 3.3.2). In order to communicate with each other, they

Solution Architecture

make use of the observer pattern, by subscribing to other modules from which they wish to be notified when new information is available. This is illustrated in figure 3.2.

This notification system is possible through usage of interfaces, both for the trackers/providers of information and the observers of other modules. Below is one example of a tracker interface and one of an observer interface.

```
1  public interface IDisplacementTracker {
2      /**
3       * Starts the displacement tracker, by registering in the appropriate modules
4       * . This call enables notifications to the registered observers.
5       */
6      void start();
7
8      /**
9       * Stops the displacement tracker, by unregistering in the appropriate
10      * modules. This call disables notifications to the registered observers.
11      */
12      void stop();
13
14      /**
15       * Subscribes an observer to notifications from the displacement tracker.
16       * @param obs observer to be registered
17       */
18      void register(IDisplacementObserver obs);
19
20      /**
21       * Subscribes an observer to notifications from the displacement tracker.
22       * @param obs observer to be unregistered
23       */
24      void unregister(IDisplacementObserver obs);
25
26      /**
27       * Sends a notification to all the subscribed observers with a new
28       * displacement vector.
29       * @param displacement displacement vector to be announced
30       */
31      void notifyObservers(Displacement3f displacement);
32
33      /**
34       * Defines the current heading used in displacement tracker to a user-defined
35       * value.
36       * @param heading the new heading value to be used
37       */
38      void setCurrentHeadingValue(float heading);
39  }
```

Listing 3.1: Example of a tracker interface (IDisplacementTracker).

```

1  public interface IDisplacementObserver {
2      /**
3       * Method invoked whenever a new displacement vector is computed.
4       * @param displacement computed displacement vector
5       */
6       void onNewDisplacementReceived(Displacement3f displacement);
7  }

```

Listing 3.2: Example of an observer interface (IDisplacementObserver).

All of these modules are contained within a system (BikeNav System), which uses the singleton pattern: a single, shared instance of the system. The purpose is twofold. This ensures only one system is actually being used (despite being able to have multiple implementations of it), and allows direct access to any modules. This last point may seem a contradictory measure against the observer pattern, but in fact is very useful to provide updated information calculated in "higher" modules. For instance, Position Tracker forces new GPS-obtained headings into Heading Tracker, to keep it updated as a starting point for subsequent calculations.

3.3.1 Providers

In the library context, providers are modules which represent sensors that insert new data into the system. Although they represent physical entities, having this extra level of abstraction allows providers to both connect to real, physical hardware, gathering data in real-time, or just inject pre-collected data from files, allowing for faster and automatic testing of new algorithms to improve the system's accuracy.

3.3.1.1 GPS Provider

This is the module which outputs locations from a GPS system (although the interface can be used to connect to other GNSS systems such as GLONASS with little or no change). Each output object contains: timestamp of the gathered data; latitude, longitude and altitude of the receiver; bearing relative to the North Pole and the accuracy of the current position.

3.3.1.2 Odometer Provider

This module outputs data from an external odometer (once again, this level of abstraction allows for usage of any technology by implementing the interface methods), in which each object provided must contain: timestamp of the wheel event, the instant speed, and the wheel radius of the bicycle being used. However, since many odometers also have attached a cadence sensor, the data structure is also prepared to receive such information (crank rotation timestamp and crank rotations per minute), although it is currently not being used in the system.

3.3.1.3 Barometer Provider

This provider outputs pressure readings from a barometer, in order to posteriorly calculate altitude by using this data (since we can correlate these two quantities).

3.3.1.4 Sensor Fusion Provider

The module combines the data coming from the accelerometer provider (which gives acceleration readings in the three axis of the smartphone), the gyroscope provider (which gives angular speed values of rotations in the three axis of the smartphone) and the magnetometer provider (which provides the three coordinates of the vector pointing to the magnetic north using the smartphone as the origin of the referential) to, through integration (simple or double, as explained in chapter 2), project the data given in the body frame to the global frame (the Earth coordinates). This module was already implemented by Fraunhofer and used in the development of the project as a black box.

3.3.2 Trackers

Trackers are modules which, contrarily to providers, do not insert new data into the system: instead, they use and combine data which is already present to calculate relevant quantities relative to the bicycle movement. This is made possible by having each module subscribed to providers or other trackers which output relevant information for them. Such event-driven architecture, besides improving system performance and responsibility (by avoiding polling techniques), allows for greater code extensibility, since modules can be replaced on-the-fly between program runs, without needing to change the remaining code.

3.3.2.1 Altitude Tracker

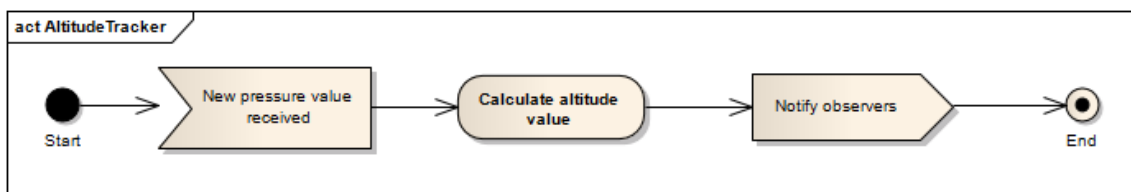


Figure 3.3: Overview of Altitude Tracker's flow of information.

This module is responsible for computing and notifying its observers about the altitudes correspondent the pressures registered in the system (by registering itself in the Barometer Provider module).

3.3.2.2 Heading Tracker

The Heading tracker module is responsible for, using inertial data provided by the Sensor Fusion module, calculate the heading of the bicycle. In order to achieve this using the smartphone sensors,

several strategies can be employed. In this version of the library, two of them were implemented: the first uses gyroscope information; the second, magnetometer information.

Gyroscope-based Heading Tracker As previously explained in chapter 2, gyroscope measures the angular velocity around its three rotational axis. Since data provided by the sensor fusion module already comes projected in the Earth's referential (global frame), we can directly calculate the rotation around the Earth's Z-axis (X-axis corresponds to latitude, Y-axis corresponds to longitude and Z-axis corresponds to altitude). Such can be obtained by calculating the time interval between the last received sensor fusion event and the current one (since both contain their respective timestamps) and then multiplying them by the angular speed received. Having this information, we can integrate this angular displacement to the last heading calculated, resulting in the current heading. Simple method, only requiring a good initial heading estimate (provided, in real usage, by GPS) in order to properly work.

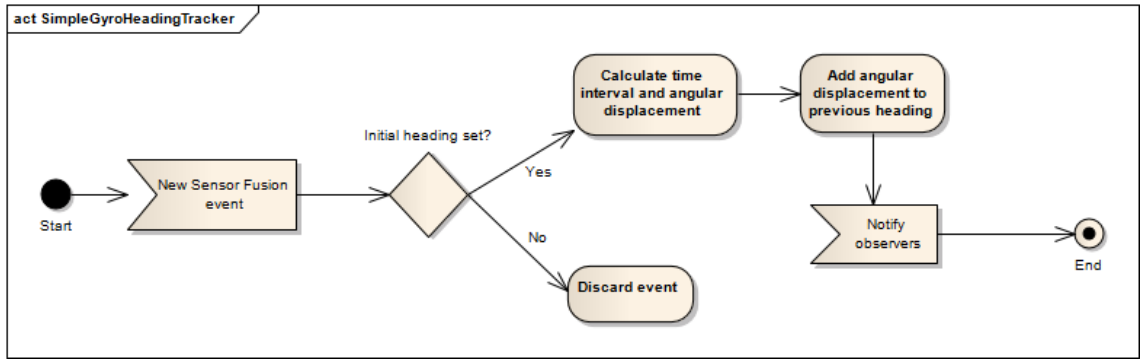


Figure 3.4: Overview of Gyroscope-based Heading Tracker's flow of information.

Magnetometer-based Heading Tracker The sensor fusion library provides events which contain magnetometer data, consisting in three-dimensional vectors pointing towards the North Pole. In order to use magnetometer data, we must, at least, have two vectors pointing north in two separate time instants, as shown on figure 3.5.

Having these two vectors, we can trivially calculate the angle between them, using the dot product:

$$\vec{a} \cdot \vec{b} = a_1 \cdot b_1 + a_2 \cdot b_2 \quad (3.1)$$

$$\cos^{-1} \theta = \frac{||\vec{a}|| \times ||\vec{b}||}{\vec{a} \cdot \vec{b}} \quad (3.2)$$

Since the θ will always be an angle between 0 and π radians, the real sign of the angle must be found, depending on the direction of travel. For that, the cross product between the two vectors pointing north is computed. If the resulting vector points in the opposite direction to gravity, then the angle should be multiplied by -1 .

Solution Architecture

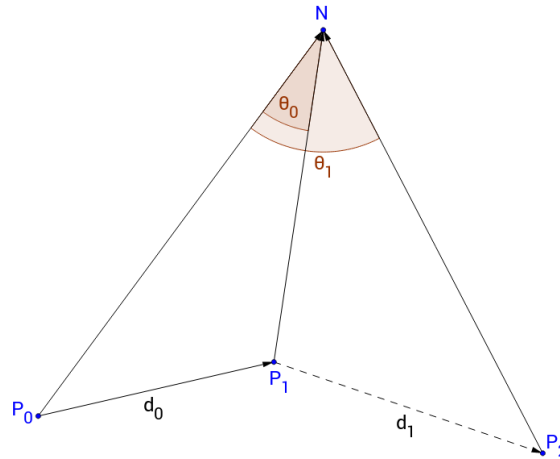


Figure 3.5: Representation of magnetometer vectors and calculated heading angles. Angles θ_n represent the angle between the vectors pointing north (heading) (N) in points P_n . D_n vectors represent the bicycle trajectory.

Using the magnetometer instead of the gyroscope to compute heading has the advantage of not being dependent from any other external system (GPS, for example) to provide an initial heading to the tracker, and since it always computes an absolute heading value, it does not accumulate drift like the gyroscope. However, it may suffer from strong calibration problems (may not point to the right north) due to a variety of reasons, such as the presence of some specific materials (like steel or iron). To amend this calibration problem, an assumption is made saying the first heading calculated (a few milliseconds) after the last GPS position is obtained is reasonably similar to the bearing provided by that same GPS data. Based on this, the difference between the two headings is computed and used as a corrective factor for the subsequent heading calculations until a valid GPS fix is obtained again. This also has the side effect of solving the magnetic declination problem (the difference between magnetic and geographical North), since the values provided by the Sensor Fusion Provider are in relation to the magnetic North.

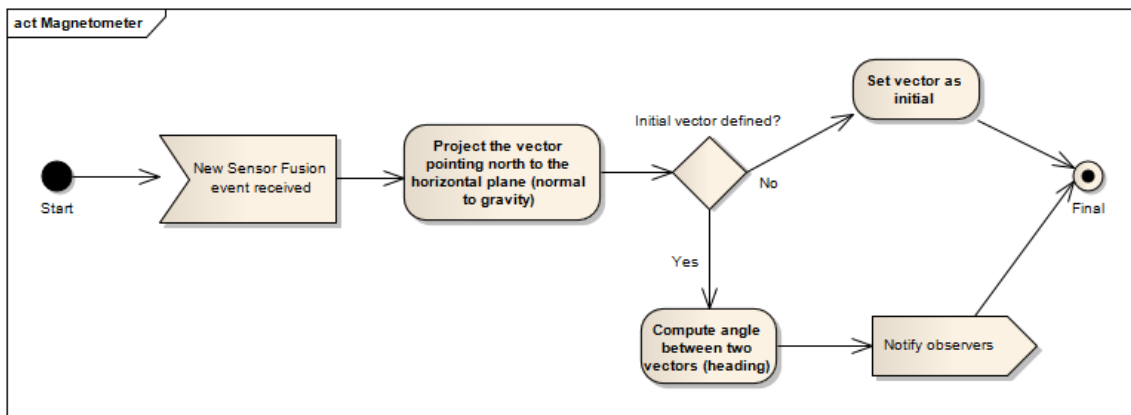


Figure 3.6: Overview of Magnetometer-based Heading Tracker's flow of information.

3.3.2.3 Revolution Tracker

Revolution Tracker is the module responsible for interpret data from Odometer Provider. At each wheel revolution event received, it computes the distance using the speed provided by the odometer multiplied by the time interval between the last received event timestamp and the current one.

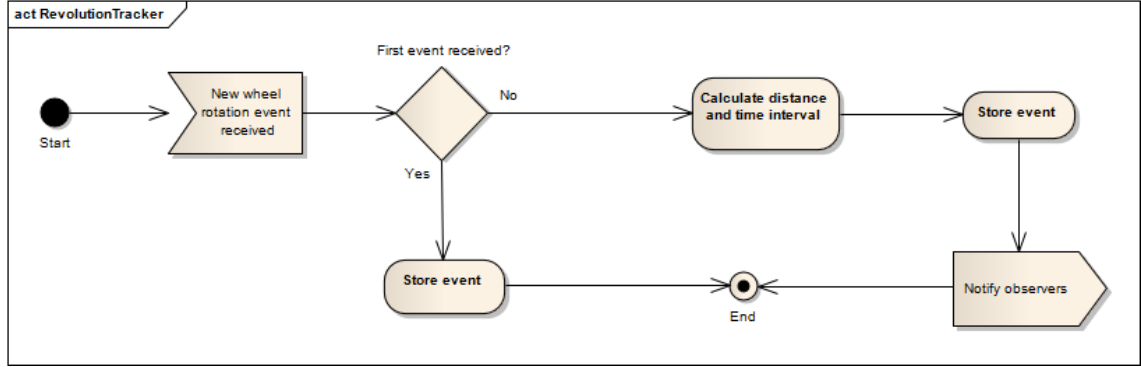


Figure 3.7: Overview of Revolution Tracker's flow of information.

3.3.2.4 Displacement Tracker

This module subscribes itself to three other trackers: it receives altitude values from Altitude Tracker, distance values computed by Revolution Tracker and heading values obtained from Heading Tracker.

When a new heading value is received, it is stored in a buffer of heading values.

When a new altitude value is received, it is stored as the current altitude value, waiting for a new distance value to be received in order to be used.

Displacement calculations (and consequently, notifications to observers) are only triggered when new distance values are received. At this point, the collected headings since the last notification received are combined into an average value. Using this value, we can create the first two coordinates of the displacement vector as follows:

$$x = d \cdot \cos \theta \quad (3.3)$$

$$y = d \cdot \sin \theta \quad (3.4)$$

where d is the distance, θ is the heading angle (in radians). For the z-axis displacement, the difference between the last used altitude and the current one is computed.

Solution Architecture

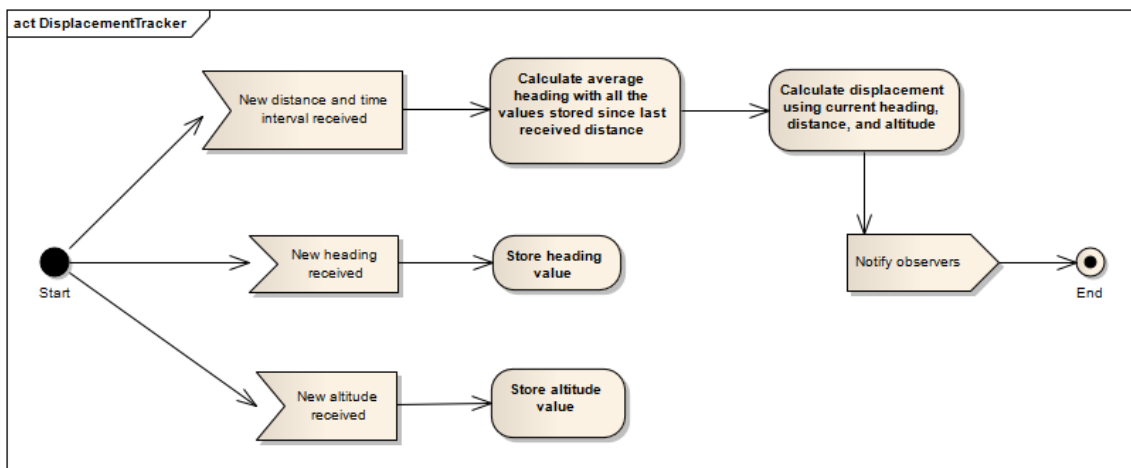


Figure 3.8: Overview of Displacement Tracker's flow of information.

3.3.2.5 Location Tracker

Location Tracker, as its name indicates, keeps track of the positions sent by the GPS Provider module, informing, if needed, its observers. In order to eliminate useless points, a simple filter is implemented which removes the ones who are not, at least, at more than 1 meter of distance from the previously received point.

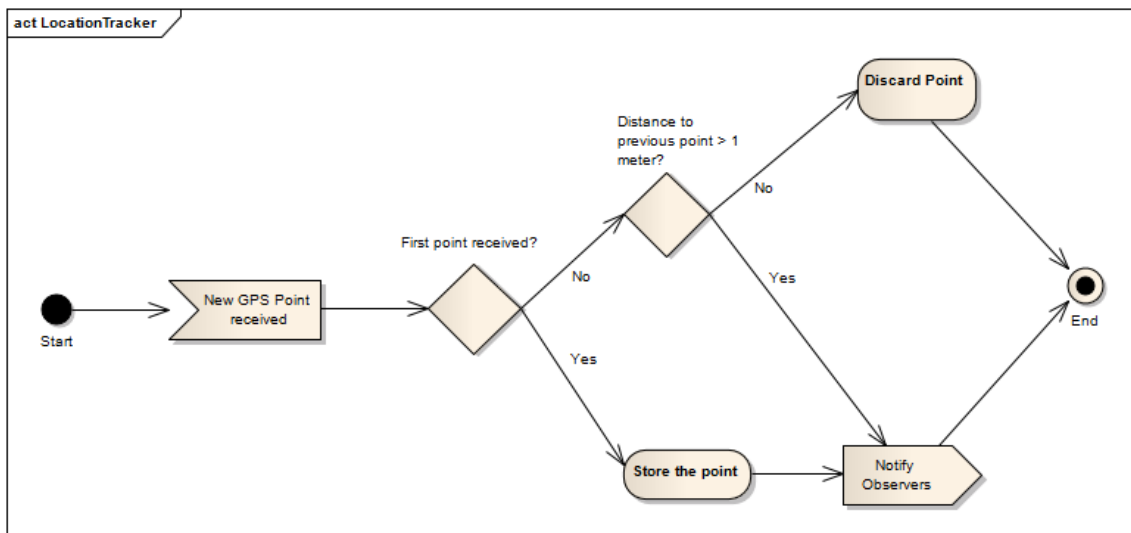


Figure 3.9: Overview of Location Tracker's flow of information.

3.3.2.6 Position Tracker

Position Tracker is the module which aggregates both the information available from the INS, represented by displacement vectors sent by Displacement Tracker as well as the information available from the GPS, through the Location Tracker.

Solution Architecture

Received displacement vectors are stored in a buffer structure, waiting to be used if a GPS failure is detected. Without this mechanism, the system's accuracy would fall, since until that GPS failure is detected, a large distance can be travelled (for example, at 7 m/s, which corresponds to 25.2 km/h, if a failure is only detected after 5 seconds, 35 meters would be travelled without any data available, since it would be erroneously discarded). In order to integrate displacements, it is needed to convert the Cartesian coordinates of the vector into spherical ones - assuming the Earth is a sphere will not increase the error significantly in this case. The expressions used to compute the increments to be added to the latitude and longitude of the previous position are as follows:

$$\Delta\Phi = \frac{x}{R} \quad (3.5)$$

$$\Delta\lambda = \frac{y}{R \cdot \cos \Phi_0} \quad (3.6)$$

where $\Delta\Phi$ and $\Delta\lambda$ are the increments (in radians), respectively, of latitude and longitude, x and y are the coordinates of the displacement vector, R is the Earth's radius and Φ_0 is the latitude (in radians) of the last position. Due to the nature of spherical coordinates, z-axis displacement (altitude) can simply be added to the previous one.

When a GPS location is received from Location Tracker, an internal signal is triggered to indicate the existence of a valid GPS fix. Such data, in this system, has always priority over the one provided by the INS modules.

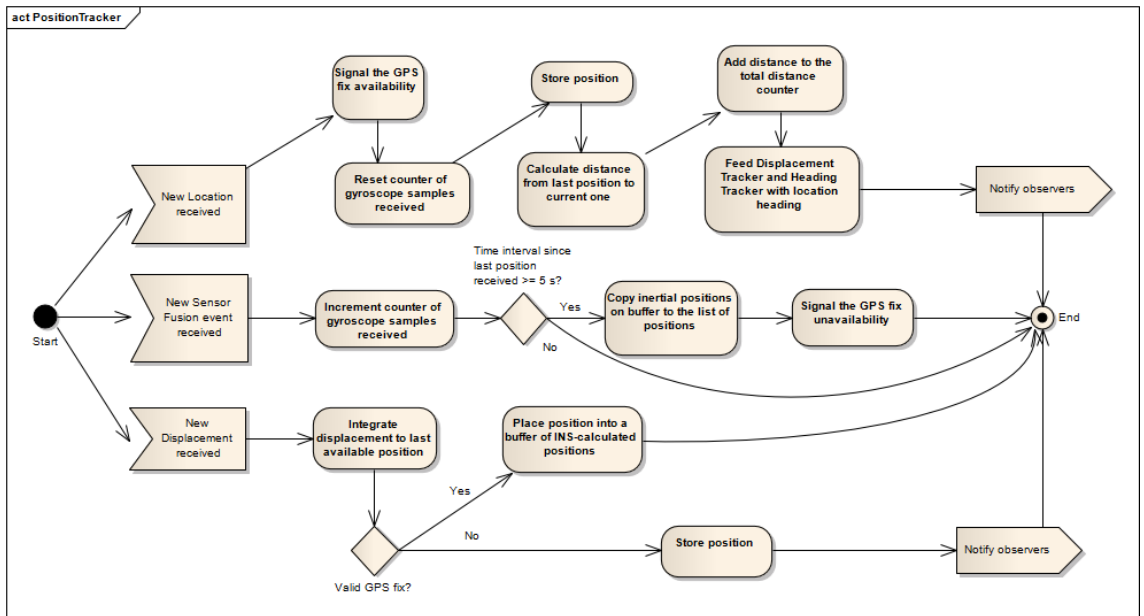


Figure 3.10: Overview of Position Tracker's flow of information.

Furthermore, the Position Tracker also observes the Sensor Fusion module, in order to keep track of time (as will be explained in chapter 4, the library may not be executed in real time). If a certain threshold time is surpassed, the tracker triggers internally a signal which indicates that no

Solution Architecture

valid GPS fix is currently available, thus enabling the acceptance of INS displacements received and promoting the stored buffer of positions inertially calculated since the last GPS position was available to valid positions of the cyclist's ride.

Chapter 4

Data Collection and Results

In order to ensure the system produced accurate results, a complete and comprehensive test process was needed. For that reason, a framework was designed and developed in order to integrate the development process, which was broke down to several iterations. This approach ensured that a software version was always available to compile and run, and that changes could be immediately tested, reducing the risk of bugs with unknown cause. In this chapter, the whole test process will be described, and the results obtain with those same tests will be presented.

4.1 Test Process Overview

As illustrated on figure 4.1, the development started by creating a good test framework for injecting data into the system and produce test reports: since the library's information providers are created by implementing interfaces, this means that any stream of information can be used to input data into the system, as long as it implements the necessary methods. This behaviour greatly accelerates testing, by reusing pre-collected data and simulate bike rides with several kilometres in seconds.

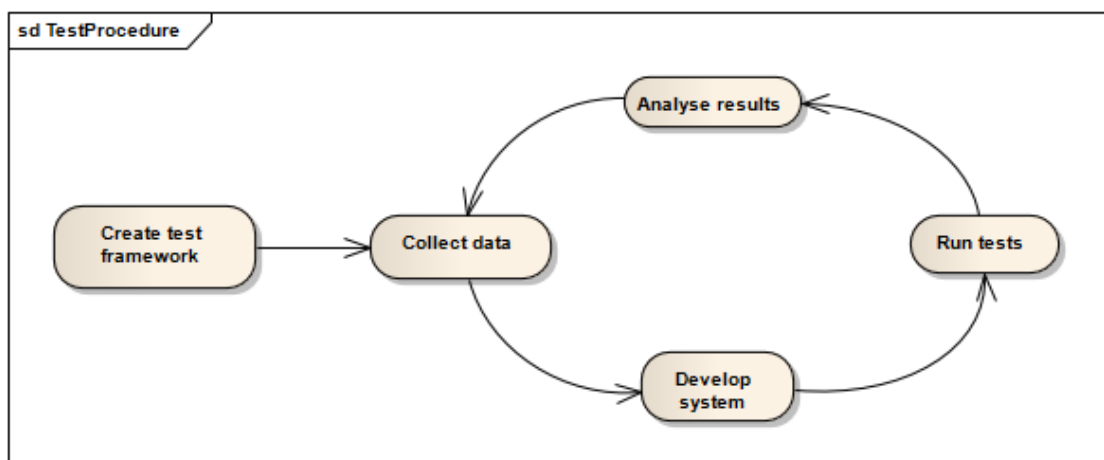
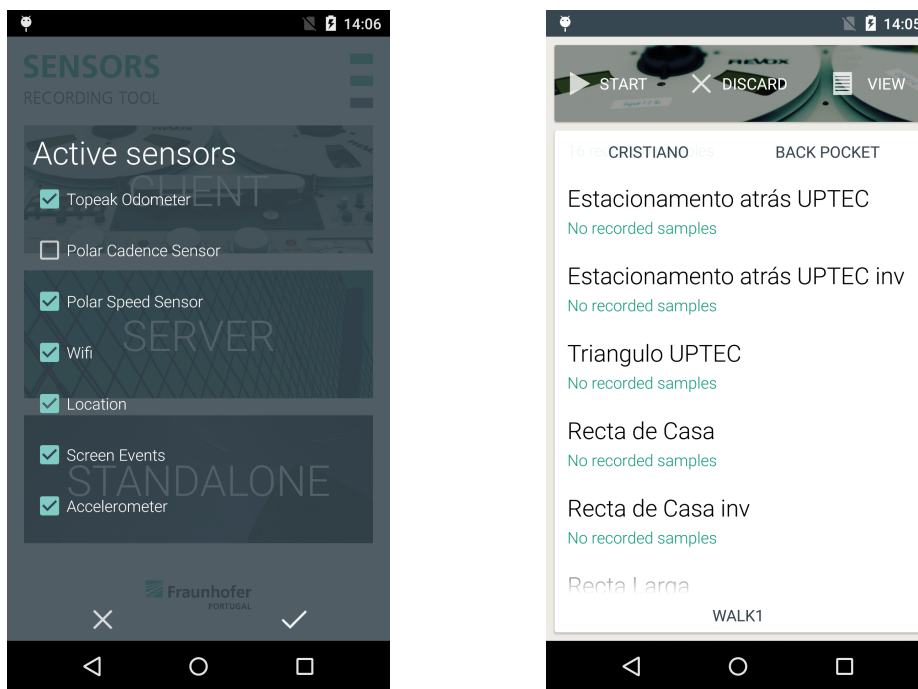


Figure 4.1: Overview of the iterative development process.

After that, an existing data recording application, developed by Fraunhofer, was extended to match the project requirements. With that application, a reasonable amount of data was collected during bike rides. With that data already stored and organized, the process continued through a small development of the system, correspondent to an iteration (*one change at a time* strategy). The test framework then used the new version of the software to produce new test reports that were posteriorly analysed to verify if an improvement had been achieved.

4.2 Recorder Application

To collect the information needed to use in test runs, an application developed by Fraunhofer, called *Recorder* was used. This highly configurable tool can collect raw data from all the available sensors of the smartphone on demand - the programmer can start and stop the recording manually and can choose from which sensors should data be stored (see figure 4.2a). The recording is always associated with a user-created route (see figure 4.2b), providing better data organization.



(a) The sensor selection menu of the recording application.

(b) The route selection menu of the recording application.

Figure 4.2: The Recorder application, used to collect data from bike rides.

Since the developed system consists in usage of smartphone sensors together with odometer data, two speed odometers were mounted in two different bicycles: in a mountain bike with 26" wheels, a Polar Speed Sensor was mounted (see figure 4.3a), and on a road bike with 28" wheels, a Topeak PanoBike Speed and Cadence Sensor was set in place. Both sensors communicated without wires, using the Bluetooth Low Energy (BLE, or Bluetooth 4.0) technology. The choice

was not innocent: the Polar is a high-end sensor, and Topeak is an entry-level one (in the range of available similar sensors).



(a) Polar Speed Sensor mounted on the test bicycle with 26" wheels.



(b) Topeak PanoBike Speed and Cadence Sensor mounted on the test bicycle with 28" wheels.

Figure 4.3: The speed sensors, mounted on the test bicycles.

To allow recording of data from these sensors, the existing *Recorder* code was forked and extended to establish BLE connections with the devices, by subscribing to the CSC (Cycling Speed and Cadence) measurements characteristic events provided by the CSC service, as defined in the Bluetooth 4.0 Reference Manual. Due to the lack of stability of the BLE stack in Android versions 4.3 and 4.4, this version of the *Recorder* app is recommended to be used in 5.0+ Android versions.¹ However, it is compatible with Android 4.3+ versions.

Each recorded ride generates a file for each of the activated sensors, with entries ordered by their arrival timestamp. Besides these individual files, there are two relevant special files: an *imu.txt*, which combines all the readings obtained from the accelerometer, gyroscope and magnetometer, and an *all.txt*, which combines all of the readings from every recorded sensor. In these files, where different sensors were mixed, an identifier was added to each one of the readings, to make possible the identification of the respective sensor (see listing 4.1).

```

1  ...
2  A,10840900633505,-1.2270966,5.5317993,-6.9472046,3
3  G,10840900633505,0.018951416,-0.10359192,-0.26246643,3
4  A,10840904844931,-1.1937714,6.074524,-6.7543945,3
5  G,10840904844931,-0.032318115,-0.10359192,-0.2464447,3
6  M,10840899962118,12.754822,-5.026245,40.96985,3
7  A,10840909819296,-1.2604218,6.5386963,-6.763916,3
8  G,10840909819296,-0.06329346,-0.11747742,-0.23149109,3
9  ...

```

Listing 4.1: Sample of a recorded *imu.txt* file. Identifiers A, G and M stand, respectively, for accelerometer, gyroscope and magnetometer readings.

¹ See <https://code.google.com/p/android/issues/detail?id=5838> for more information.

4.3 BikeNav Library Testing

Since the BikeNav library is the core of the project, it was crucial to check the accuracy of the calculated positions. To do that, a test framework was developed using Robolectric as its main component: a library which allows Android unit testing in a regular JVM, without any emulator, by mocking a large portion of the Android SDK. Using this approach, the collected data could be used to mock bicycle rides.

Figure 4.4 gives an overview of the test framework structure. It is divided in two big phases: loading and injection of data; exportation of results.

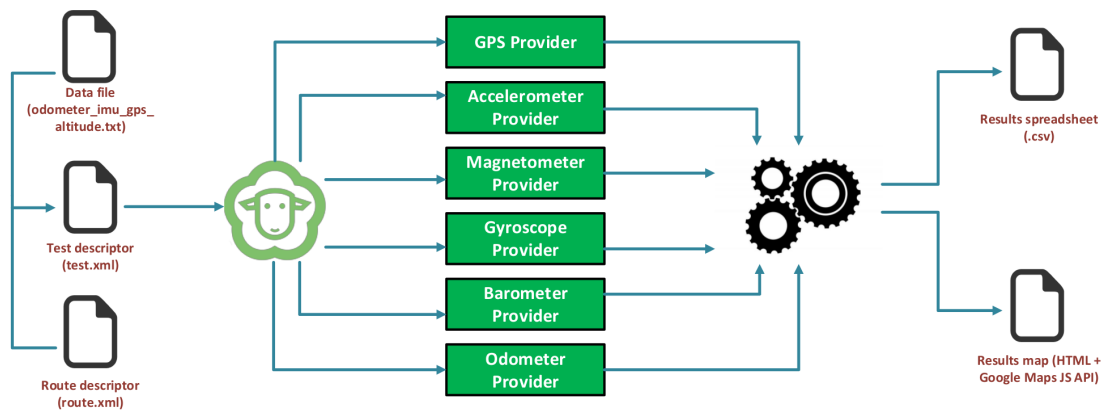


Figure 4.4: High-level overview of the test framework structure.

4.3.1 Data loading and injection

Each of the user-defined routes had two important test files associated: the route descriptor and the test descriptor. The first one contains important data about the route: its name, waypoints and initial altitude (see listing 4.2). Since, as it is explained below, some tests are run without using GPS at all, it becomes important to define these points in order to have initial information to feed the system. The initial heading, key to Heading Tracker initialization, is calculated by using the coordinates of the first two points².

These XML files were created based on information extracted from Google: the routes were planned and marked using the "My Maps" functionality of Google Maps, then exported to KMZ format (zipped Keyhole Markup Language, or KML, files³). From these KML files, for each route, the waypoint coordinates were extracted.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <route name="Recta de Casa">
3 <point id="start" latitude="40.6324051" longitude="-8.6223042" altitude="35.718"/>
4 <point latitude="40.6325944" longitude="-8.6225536"/>

```

²The formula used to calculate heading is described in <http://williams.best.vwh.net/avform.htm#CrS>.

³See <https://developers.google.com/kml/documentation/kmzarchives> for more information.

Data Collection and Results

```
5 <point latitude="40.6330768" longitude="-8.6230633"/>
6 <point latitude="40.6333944" longitude="-8.6233959"/>
7 ...
8 <point latitude="40.6371335" longitude="-8.626973899999999"/>
9 <point id="end" latitude="40.6375874" longitude="-8.6275935"/>
10 </route>
```

Listing 4.2: Sample route.xml file.

The second XML contains the description of the test itself: it specifies which route will be loaded; whether to use the route descriptor initial point or to use the logged GPS information. But, most importantly, especially for medium and big rides, it defines the cuts which will be made to the GPS signal, which serve to fully rely (and therefore, test) on the INS system modules. These GPS cuts can be in certain, user-defined, distance intervals (in meters), as shown in listing 4.3, or they can be random, in which the framework, when loading the test, will create random intervals in which the GPS signal will not be available. One final option is to completely cut the GPS signal, relying solely on inertial data.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <test>
3   <route name="Volta Media" />
4   <settings>
5     <initial_position type="route" />
6     <final_position type="route" />
7   </settings>
8   <nogps type="distance">
9     <event start="1800" end="3000"/>
10    <event start="3900" end="4500"/>
11    <event start="5100" end="7000"/>
12  </nogps>
13 </test>
```

Listing 4.3: Sample test.xml file.

It is important to note these cuts are done in Location Tracker, not on GPS Provider itself; this approach simplifies the collection of data for posterior statistical analysis, since data is always injected to the system, regardless of being used or not.

After loading these two XML files, the test then loads the corresponding data for rides which travelled that route. Since raw data files (all.txt) can easily reach hundreds of MB in size, a Ruby script was created, which extracts and copies the relevant data (GPS, accelerometer, gyroscope, magnetometer, barometer) into a separate file, *odometer_imu_gps_altitude.txt*.

Segments vs. Tracks Given the importance of these two concepts in the test definitions, it is important to formalize the differences between them. Both are routes, with arbitrary initial and final points. The difference between them is distance: segments are routes that are less than

1000 meters long; tracks are 1000+ meters long. This distance has a reason: segments must be small, so several records can be done quickly, but some of them must also be large enough to detect possible drifts, especially in heading. This influences their criteriums for data collection: segments do not need valid GPS data in recordings; since they are small, GPS data may not be accurate enough to provide reliable testing data; instead, their route descriptor should contain all the needed waypoints. Track recordings, however, are not useful without valid GPS data attached to them, due to their extension.

4.3.2 Exportation of Results

The system exports some relevant collected and calculated data, to allow results analysis in both a visual and detailed way. Visually, for each test, it creates an HTML file, with CSS and JS embedded to display a map with: the "perfect" route, as drawn with the Google My Maps platform (for segments), the processed GPS data (for tracks), and the resulting positions obtained through Position Tracker (which are GPS+INS and depend on the GPS signal cuts made). An example of such map can be seen on figure [4.5](#)

In a more detailed way, it creates two spreadsheets: one, *gps.csv*, which contains all the positions observed by Position Tracker from Location Tracker (GPS positions) and *ins.csv*, which contains all of the resulting final positions from the system, which is the GPS + INS combination.

4.4 Test Runs and Obtained Results

With the test framework already working, several tests were designed to assess the accuracy of the system. Two big quantities were tested: the travelled distance (and, per consequence, the travelling speed), provided by the attached odometer, and direction of travel, provided by the inertial sensors - the heading.

4.4.1 Odometer distance validation

As already explained in chapter [3](#), the odometer was introduced in the system to provide more accurate readings, since the noise in values provided by the accelerometer is very high. To validate the odometer values, small segments were travelled with the bike, and distances to the "ideal" ones were compared, as shown in table [4.1](#).

As it can be seen, mean error between the real distance - as traced with the Google My Maps platform and the calculated distance varies between 9.78 meters and 105.59 meters. However, this difference is not explained due to an error accumulation, since there is not any visible relationship between mean distance and mean error.

A curious finding is that, although it was expected to have higher mean calculated distance values than the real ones, since it is very difficult to keep the bicycle always in a straight line - there is a need to avoid traffic, obstacles and even natural body movements make it difficult to keep a steady position (even for very experienced cyclists), the means are consistently lower than

the reference values. This can be explained with how the odometers work. In order to save energy, these sensors stop emitting BLE signal, entering a latency state. This means that, since waking up these sensors is not instantaneous, there will always exist a time interval where no events will be detected, and consequently, calculated distances will be smaller than expected. In real-time usage (i.e, directly processing data provided by real hardware sensors, not previously collected one), such problem will not exist, since sensors will be already awake with the bicycle motion - even if the system is not using that information due to GPS signal availability.

Table 4.1: Results of distance calculation with odometer. Real distance, in this context, is the one measured with My Maps platform. Mean distance is the mean of all distances calculated for that specific segment (using different data samples), and the presented standard deviation is referent to this mean. Mean error is the mean of all differences to the real final position from the calculated one. Min and Max distance are the minimum and maximum calculated segment distances.

Segment Name	Real Distance (m)	Mean Distance (m)	Standard Deviation (m)	Mean Error (m)	Mean Error / 100m (m)	Max Distance (m)	Min Distance (m)
Triângulo UPTEC	279	268.82	1.92	9.78	3.51	270.43	266.12
Recta Larga	167	163.76	8.69	3.43	2.05	172.44	141.68
Recta Larga (inverse)	167	166.35	2.74	0.32	0.19	169.85	160.68
Recta Rua João Calisto	117	112.17	1.08	4.75	4.06	113.13	110.84
Recta Rua João Calisto (inverse)	117	111.46	0.74	5.47	4.67	112.42	110.14
Descida Igreja	183	171.85	14.35	10.75	5.89	181.22	137.06
Subida Igreja	178	176.34	5.90	1.65	0.93	189.14	171.26
Recta de Casa	735	658.67	21.14	76.33	10.38	635.16	686.42
Estacionamento atrás UPTEC	87	77.06	1.34	10.44	11.93	78.26	75.19
Estacionamento atrás UPTEC (inverse)	87	86.33	0.00	1.16	1.33	86.33	86.33

Based on these results, one can successfully assume that odometer measures are close enough to reality for successful usage, since 70% of the test routes have mean errors inferior to 10 meters,

and even the ones who surpass this limit have substantial standard deviations related to the calculated mean distance, meaning there were, between all of those bicycle rides, at least some with a significant smaller error than the mean.

4.4.2 Heading validation

After validating the computed travelled distance, it is crucial for the reliability of the system to assess the accuracy of the computed heading value. For that purpose, tests were created to measure how reliable are the two implemented algorithms (using magnetometer-based heading and gyroscope-based heading).

To isolate the INS system (and consequently, the heading algorithms), the test was run in segments without any help from GPS at all. The initialization values of the system were defined through the route descriptors of the corresponding segments, as described in section 4.3.1. Both implemented algorithms - gyroscope-based and magnetometer-based heading - were used in test runs, to establish a comparison and highlight some of the advantages and disadvantages of each one.

Table 4.2: Results of route calculation with gyroscope-based heading. Real distance, in this context, is the one measured with My Maps platform. Mean error is the means of the distances between the final calculated positions and the final real position (and the presented standard deviation refers to this mean). Min and max error refer, respectively, to the smallest and largest distance between the calculated final position and the real one.

Route Name	Real Distance (m)	Mean Error (m)	Mean Error per 100m (m)	Standard Deviation (m)	Min Error (m)	Max Error (m)
Estacionamento atrás UPTEC	87	41.12	46.99	8.09	31.17	50.98
Triângulo UPTEC	279	66.99	24.04	3.82	61.68	70.53
Recta Larga	167	38.10	22.78	25.35	8.31	74.22
Recta Larga (inverse)	167	52.55	31.53	63.97	6.04	201.29
Descida Igreja	183	61.65	33.76	49.54	18.55	172.03
Subida Igreja	178	35.46	19.92	38.86	6.70	121.96
Recta Rua João Calisto	117	34.35	29.38	21.97	11.87	71.77
Recta Rua João Calisto (inverse)	117	11.62	9.94	6.93	5.82	26.62
Recta de Casa	735	149.87	20.39	53.53	81.24	211.86

Data Collection and Results

Starting with gyroscope-based heading, as shown in table 4.2, we can see that the error accumulated by each 100 meters remains consistent across almost all segments (the "Recta Rua João Calisto" segment has surprisingly good results, well below all other routes). However, the standard deviation related to the mean error and the difference between the minimum error and the maximum error for some routes are very intriguing.

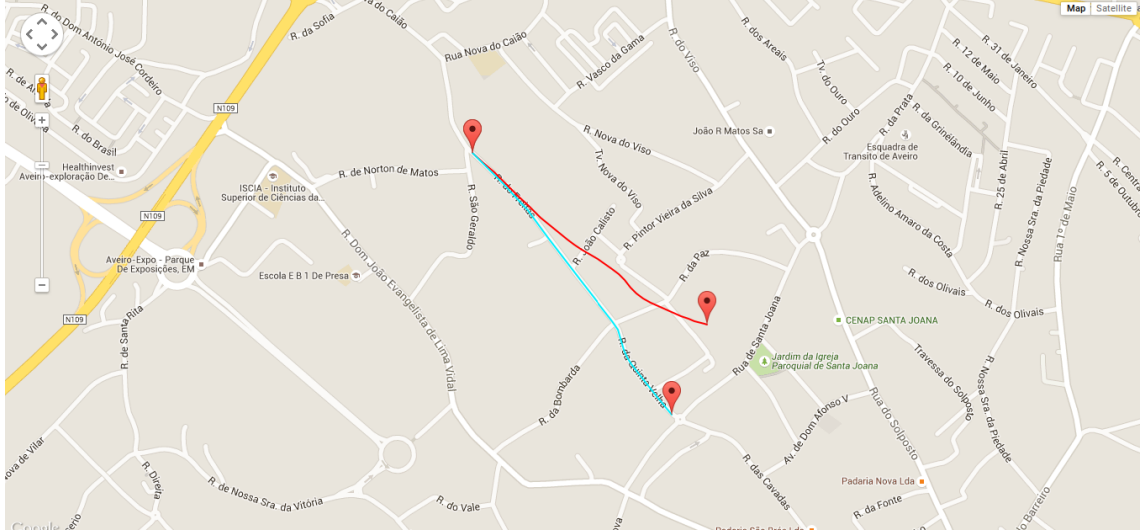


Figure 4.5: Representation in Google Maps of the 'Recta de Casa' segment with most accumulated drift using gyroscope-based heading tracker. The cyan line represents the real route, i.e., the one marked using Google My Maps. The red line represents the route calculated by the INS.

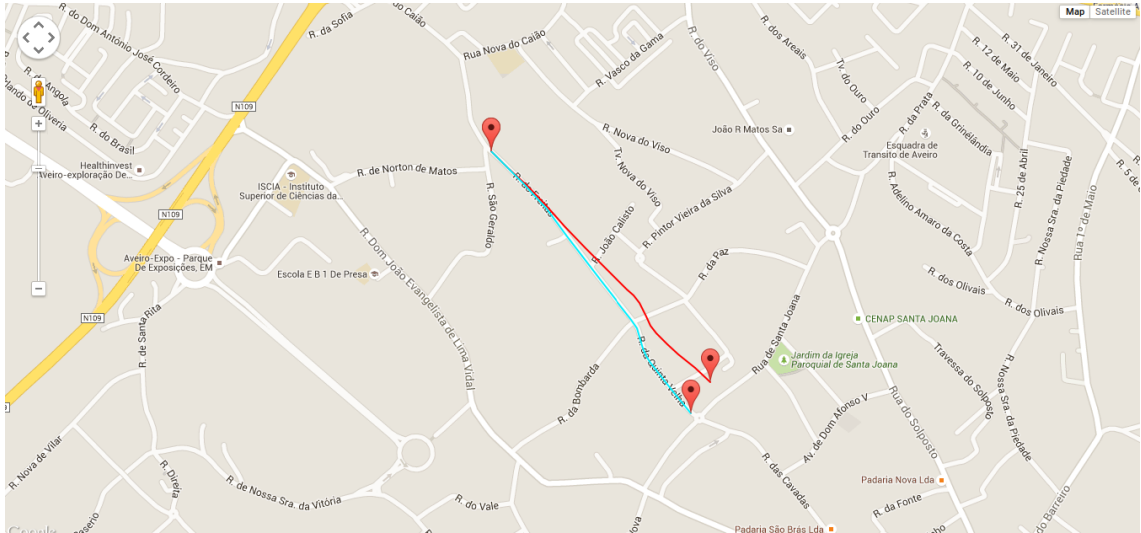


Figure 4.6: Representation in Google Maps of the 'Recta de Casa' segment with least accumulated drift using gyroscope-based heading tracker. The cyan line represents the real route, i.e., the one marked using Google My Maps. The red line represents the route calculated by the INS.

Figures 4.5 and 4.6 represent one of the cases with a large gap between the minimum and maximum final error in position. With these images, it becomes clear that, in the same conditions

Data Collection and Results

(bike and route), drift accumulation is not constant or close enough in between rides, implying there are other factors which can largely contribute to such error. One of these can be poor sensor calibration, since sensors, with the accumulation of vibrations and interferences, tend to become less and less calibrated over time. One other factor, which is also related to the first one, is the travelling speed. With the increase in speed, the bike (and, consequently, the smartphone) tends to suffer more and more violent vibrations, which directly contribute to the loss of sensor calibration. This can experimentally verified with this test framework; unfortunately, in the short duration of this dissertation, there was not possible to collect a significant amount of data which would have allowed to extract reliable conclusions from this kind of statistical analysis.

Table 4.3: Results of route calculation with magnetometer-based heading. Real distance, in this context, is the one measured with My Maps platform. Mean error is the means of the distances between the final calculated positions and the final real position (and the presented standard deviation refers to this mean). Min and max error refer, respectively, to the smallest and largest distance between the calculated final position and the real one.

Route Name	Real Distance (m)	Mean Error (m)	Mean Error in 100m (m)	Standard Deviation (m)	Min Error (m)	Max Error (m)
Estacionamento atrás UPTEC	87	50.13	57.29	14.80	33.75	69.61
Triângulo UPTEC	279	26.78	9.61	8.22	16.20	36.24
Recta Larga	167	59.12	35.36	35.79	11.89	127.10
Recta Larga (inverse)	167	109.97	65.98	60.32	18.53	228.09
Descida Igreja	183	111.64	61.14	82.80	49.82	296.16
Subida Igreja	178	54.42	30.57	52.08	14.22	177.62
Recta Rua João Calisto	117	69.02	59.03	41.32	17.09	125.70
Recta Rua João Calisto (inverse)	117	35.09	30.01	7.98	22.34	46.99
Recta de Casa	735	158.14	21.52	35.21	108.97	189.51

Using the magnetometer-based heading tracker, two things are easily visible (table 4.3): both the mean error value and the disparity of results increases when compared to the gyroscope values. This is easily explainable: between routes, there are areas with higher propensity to cause magnetic interferences, especially on residential areas, where metallic presence is a constant. In between rides of the same route, the errors can be explained, once again, through poor magnetometer

calibration.

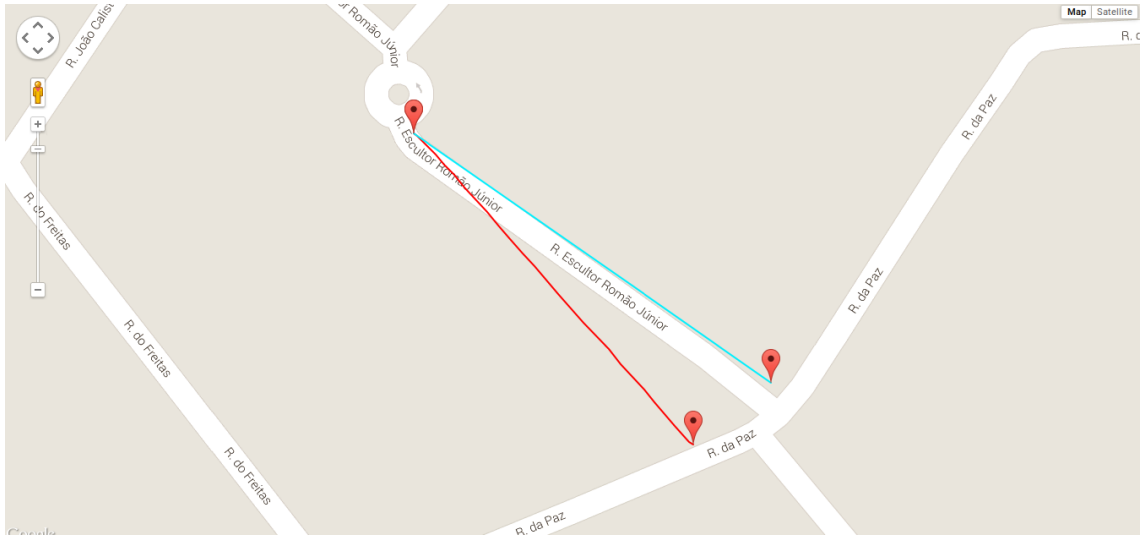


Figure 4.7: Representation in Google Maps of the 'Recta Rua João Calisto (inverse)' segment with a large error using magnetometer-based heading tracker. The cyan line represents the real route, i.e., the one marked using Google My Maps. The red line represents the route calculated by the INS.

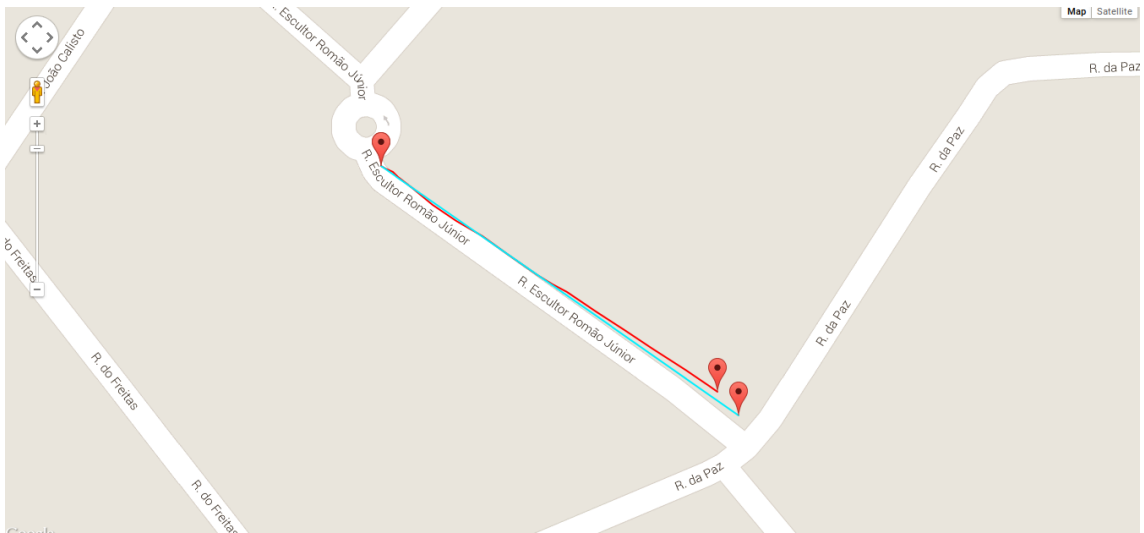


Figure 4.8: Representation in Google Maps of the 'Recta Rua João Calisto (inverse)' segment with a small error using magnetometer-based heading tracker. The cyan line represents the real route, i.e., the one marked using Google My Maps. The red line represents the route calculated by the INS.

Figures 4.7 and 4.8 illustrate the difference that poor sensor calibration can make. After calibrating the compass, the error decreased dramatically, giving, in this context, a near perfect calculation of the final position with the inertial navigation system (represented in figure 4.8). This

shows how much important is to have the magnetic compasses in the smartphone correctly calibrated. However, as explained before, this is not always enough to ensure good results with this approach, since the magnetic field sources which disturb the sensor are abundant in residential areas.

4.4.3 Tracks with GPS cuts

In the context of the Syndromic Surveillance project, described before, the Community Health Workers do not travel only 500 or 1000 meters every day, as tested in these segments; they can travel up to 25 km per day, with intermittent GPS signals. To have a more realistic approach to this usage, 3 tracks with a significant distance were ridden, and posteriorly, GPS cuts were defined to simulate the signal intermittency which will occur in the project region.

Interpreting the cut tables In this section, tables with the cuts made to the GPS signal will be presented. In those tables, Start is the distance after which the cut was detected; End is the distance after which a new GPS point was received in Position Tracker (cut finished); Distance represents the difference between the end and the start of the cut. Error represents the distance between the calculated end point of the segment and the following GPS point. It is important to note that cut intervals present in the table may drift a little from the theoretical defined ones, since distance between received GPS points is variable, i.e, a GPS point may not be received exactly at the start of the distance interval.

Interpreting the map representations For each run, a map representation it will be presented. In them, the blue line represents the received GPS data, and the red line represents the positions calculated through the INS. Although the GPS signal is projected onto the map without any cut, the INS system does not always relies on it - such is visible in periods when the red line drifts away from the GPS line.

4.4.3.1 Track 1

The first track is 9.1 km long, in the Aveiro area. Starting on "Avenida de Santa Joana", it proceeds through "Rua do Solposto", "Rua do Santo", "Rua de São Brás", "Rua dos Campinhos", "Rua das Quintas", "Rua dos Forninhos", "Rua Sociedade Musical Santa Cecília", "Estrada de São Bernardo", "Estrada de Vilar", "Avenida de Bourges", "Avenida Francisco Sá Carneiro", "Avenida Doutor Francisco do Vale Guimarães", "Rua Padre Filipe Rocha", "Rua Nossa Senhora da Vitória", "Rua das Areias de Vilar", "Rua da Patela", "Rua da Quinta Nova", "Rua Dom João Evangelista de Lima Vidal", and returns to "Avenida de Santa Joana".

This track was ridden once, and the gathered data was subject to two test runs, using different cut intervals in each of them.

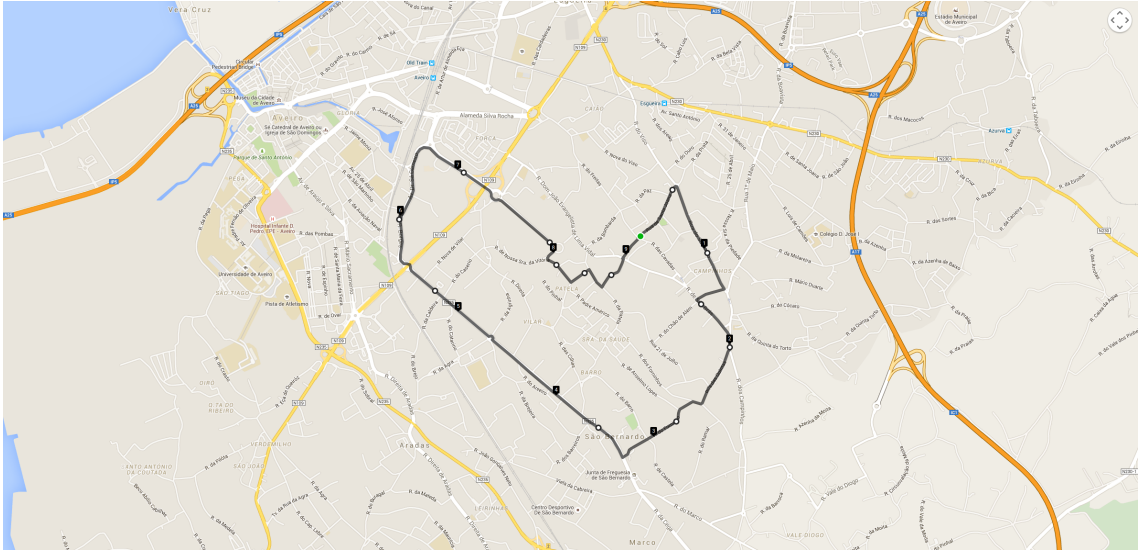


Figure 4.9: Representation in Google Maps of track 1.

First run In the first run (see tables 4.4 and 4.5), there were GPS signal cuts made between intervals [500, 1000] meters, [1400, 3500] meters, [3900, 4300] meters, [5500, 6200] meters. Comparing the values provided by each of the heading tracker algorithms, we can see that, on average, gyroscope values were much more accurate than the magnetometer values. However, the maximum error was achieved by the gyroscope in the interval [3900, 4300] meters, since it is much more susceptible to accumulate drifts and needs a good initial heading estimate in order to be effective. In this interval, a technical problem with the bicycle occurred, forcing a temporary inversion of the riding direction with both feet on the ground (figure 4.11). The cut starts in the middle of the inversion, providing an erroneous estimate to the heading tracker and causing most of the drift. As it is perceptible in figure 4.10, the magnetometer does not suffer this problem, since it uses a fixed point of reference, the magnetic north. Even if the initial estimate is wrong, it will eventually point in the right direction after a short amount of time. However, it is subject to rather "random" interferences, such as the ones in the first GPS cut, leading it to drift from the true course.

Table 4.4: Cuts made to the GPS signal in the first run of track 1, with magnetometer-based heading.

Start (m)	End (m)	Distance (m)	Error (m)	Error per 100 meters (m)
502.82	1017.72	514.9	232.77	45.2068362789
1403.02	3504.77	2101.75	135.95	6.4684191745
3902.68	4304.86	402.18	59.02	14.6750211348
5500.07	6211.26	711.19	114.95	16.1630961152

Data Collection and Results

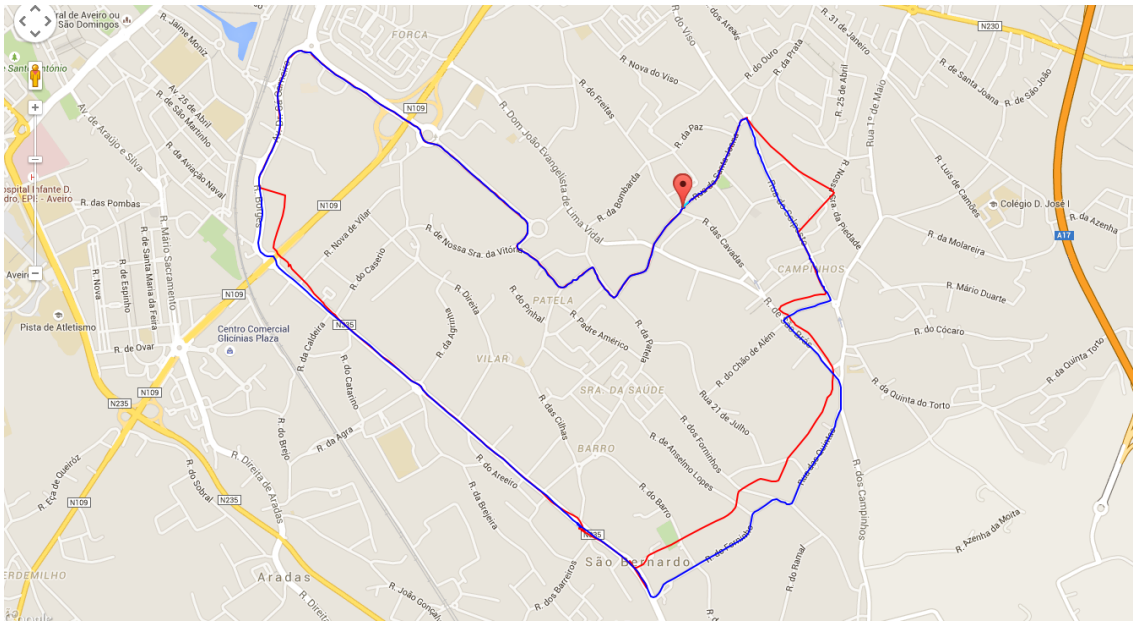


Figure 4.10: Representation in Google Maps of the first run of track 1 using magnetometer-based heading tracker.

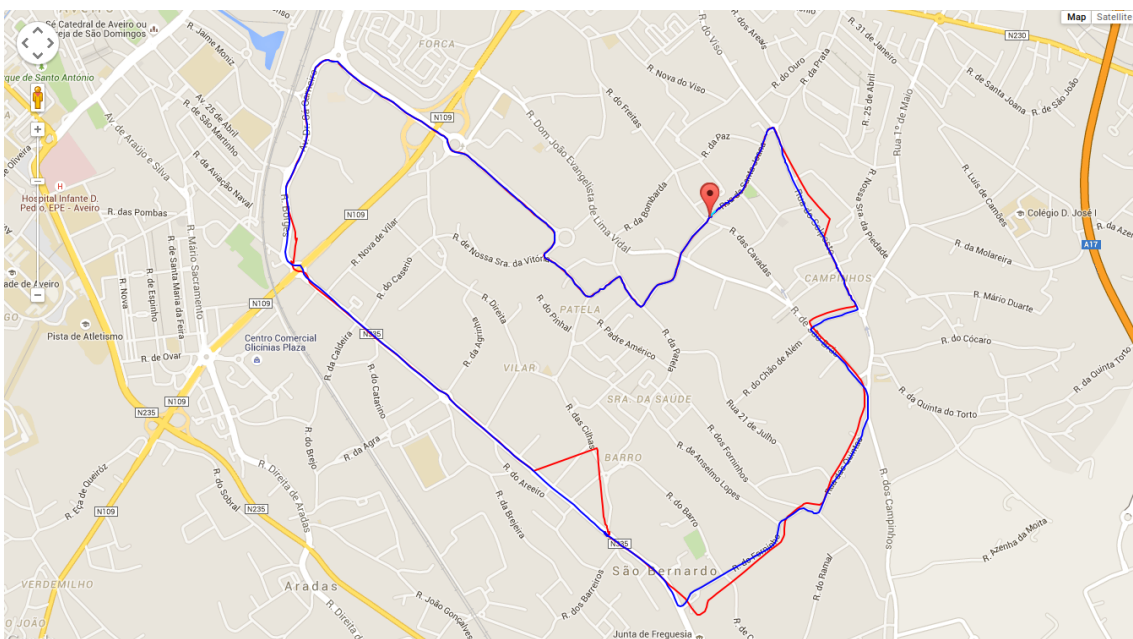


Figure 4.11: Representation in Google Maps of the first run of track 1 using gyroscope-based heading tracker.

Table 4.5: Cuts made to the GPS signal in the first run of track 1, with gyroscope-based heading.

Start	End	Distance	Error	Error per 100 meters
511.15	1001.04	489.89	82.22	16.7833595297
1405.64	3501.34	2095.7	95.11	4.5383404113
3902.49	4301.06	398.57	291.27	73.0787565547
5502.67	6200.15	697.48	88.47	12.6842346734

Second run In the second run (see tables 4.6 and 4.7), there were GPS signal cuts made between intervals [1000, 2000] meters, [3000, 3500] meters, [4000, 5500] meters and [6000, 7500] meters. Comparing once again the values obtained from both the gyroscope-based and the magnetometer-based heading trackers, once again the gyroscope produces less error than the magnetometer. The last interval is a clear example of magnetic interference, since given the initial forward estimate, the heading turned almost -180° before re-turning again to the right direction, meaning there was some sort of disturb to the magnetic field received by the smartphone sensor (see figure 4.12). In figure 4.13, in the interval [4000, 5500] meters, it is possible to see the drift accumulation on the phone, moving it slowly apart from the correct route.

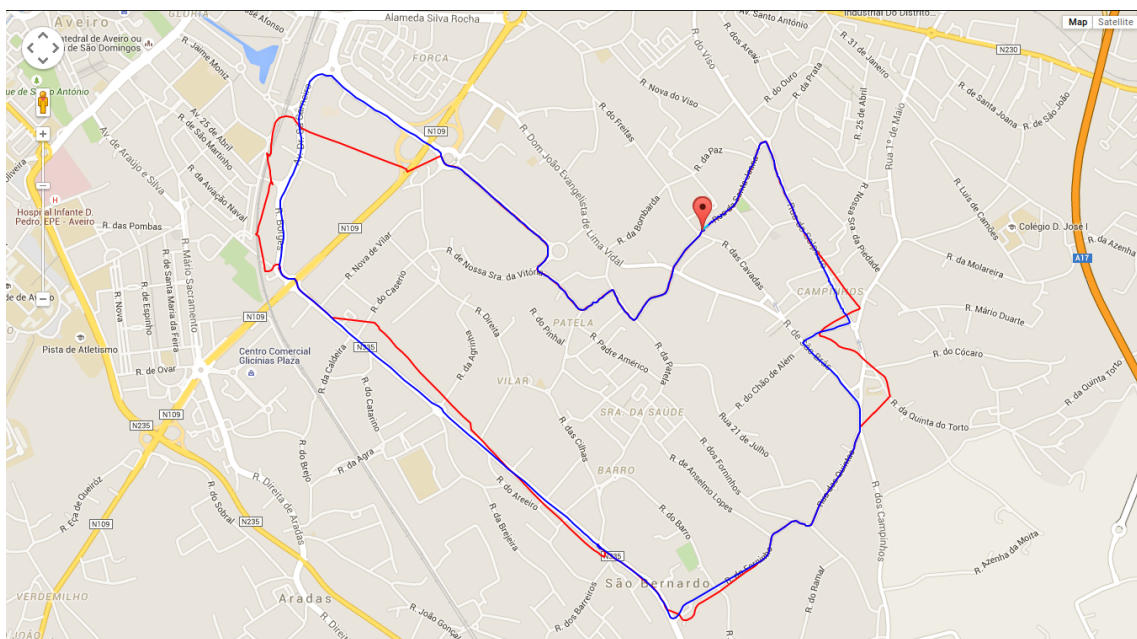


Figure 4.12: Representation in Google Maps of the second run of track 1 using magnetometer-based heading tracker.

Data Collection and Results

Table 4.6: Cuts made to the GPS signal in the second run of track 1, with magnetometer-based heading.

Start	End	Distance	Error	Error per 100 meters
1015.94	2010.63	994.69	185.29	18.6279142245
3000.1	3506.48	506.38	58.53	11.5585133694
4002.17	5507.29	1505.12	189.51	12.5910226427
6001.85	7502.26	1500.41	159.36	10.6210969002

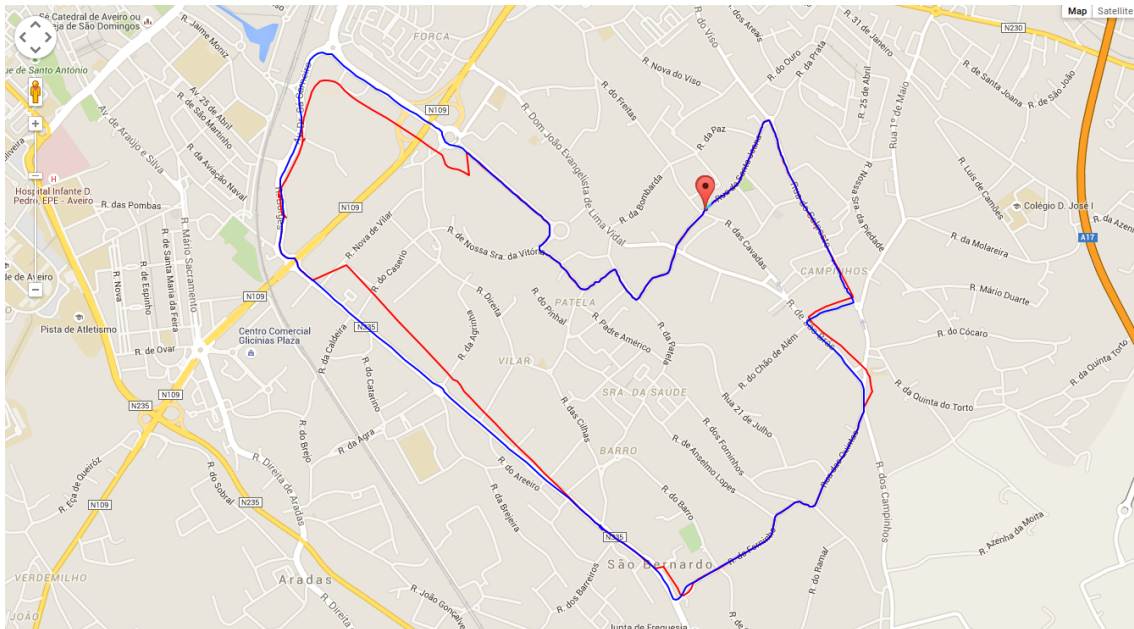


Figure 4.13: Representation in Google Maps of the second run of track 1 using gyroscope-based heading tracker.

Table 4.7: Cuts made to the GPS signal in the second run of track 1, with gyroscope-based heading.

Start	End	Distance	Error	Error per 100 meters
1050.58	2008.83	958.25	77.47	8.0845290895
3005.33	3508.26	502.93	34.37	6.8339530352
4006.3	5500.2	1493.9	157.2	10.5227926903
6007.113	7500.36	1493.247	98.49	6.5956938135

4.4.3.2 Track 2

The second track has a total extension of 5.0 km, in the Aveiro area, more concretely, in Santa Joana. Starting in "Avenida de Santa Joana", it passes through "Rua do Solposto", "Largo do Solposto", "Rua do Barreiro", "Rua da Molareira", "Rua Luís de Camões", "Rua da Azenha", "Rua Primeiro de Maio", "Rua 31 de Janeiro", "Rua Engenheiro Adelino Amaro da Costa", returning to "Avenida de Santa Joana".

Like in the first track, this one was ridden once, and the gathered data was subject to two test runs, using different cut intervals in each of them.

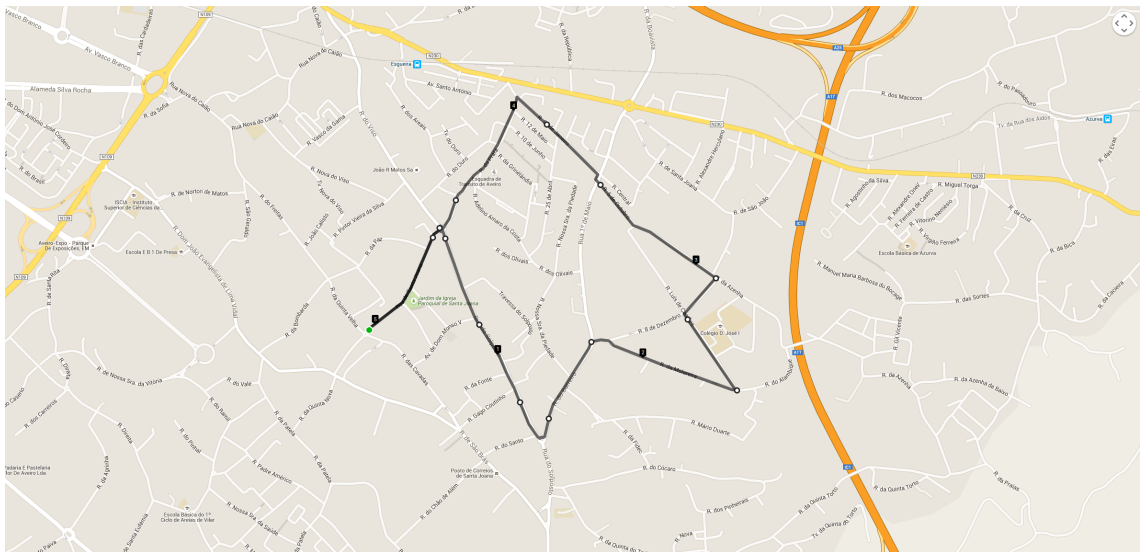


Figure 4.14: Representation in Google Maps of track 2.

First run In the first run (see tables 4.8 and 4.9), there were GPS signal cuts made between intervals [500, 1000] meters, [1400, 3500] meters, [3900, 4000] meters. Contrarily to the first track, this time, positions calculated based on magnetometer data were more accurate than the ones based on gyroscope data. Again, it becomes evident that the latter tends to be accurate in short periods of time, but after a significant period of time, it accumulates a large amount of drift, when compared to the magnetometer.

However, magnetometer also evidences a problem in this run. As already explained in chapter 3, a correction factor is calculated for two main reasons: to correct poorly calibrated magnetometers (pointing to the "wrong" north), and to compensate magnetic declination. However, if the GPS angle points towards a wrong position, correcting it only after the cut is made (see the [500, 1000] meters interval in figure 4.15), the correction angle will be wrong, and consequently, the whole calculated segment will be wrong.

Data Collection and Results

Table 4.8: Cuts made to the GPS signal in the first run of track 2, with magnetometer-based heading.

Start	End	Distance	Error	Error per 100 meters
504.36	1016.21	511.85	378.63	73.9728436065
1402.68	3506.92	2104.24	79.41	3.7738090712
3912.66	4003.06	90.4	32.91	36.4048672566

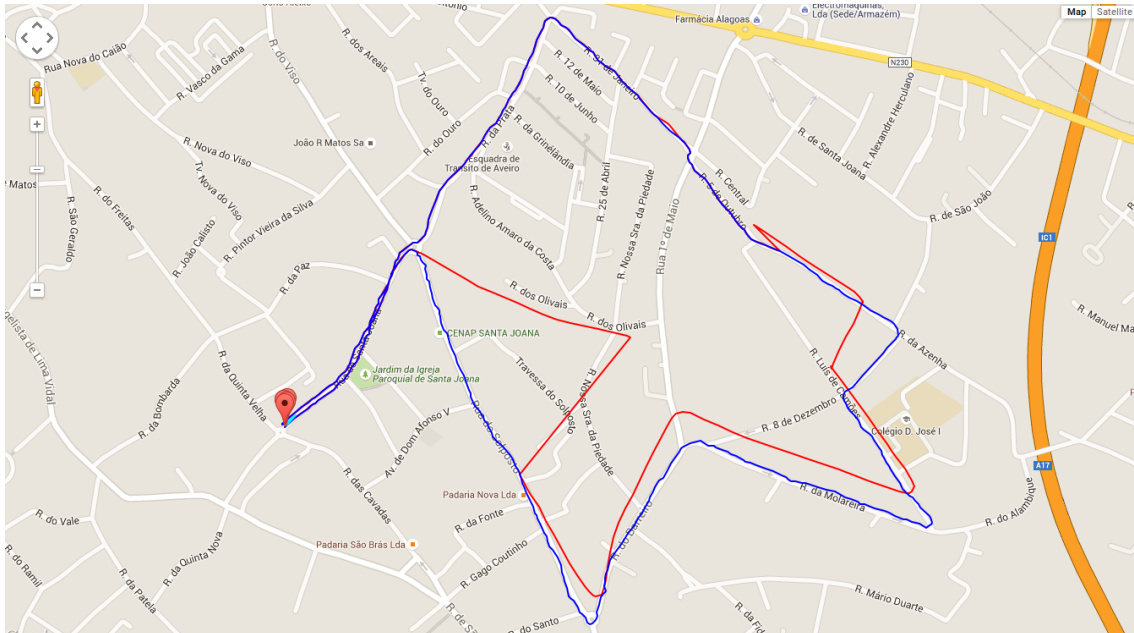


Figure 4.15: Representation in Google Maps of the first run of track 2 using magnetometer-based heading tracker.

Table 4.9: Cuts made to the GPS signal in the first run of track 2, with gyroscope-based heading.

Start	End	Distance	Error	Error per 100 meters
504.36	1016.21	511.85	434.02	84.7943733516
1467.14	3504.32	2037.18	138.55	6.8010681432
3900.54	4004.52	103.98	41.93	40.325062512

Data Collection and Results

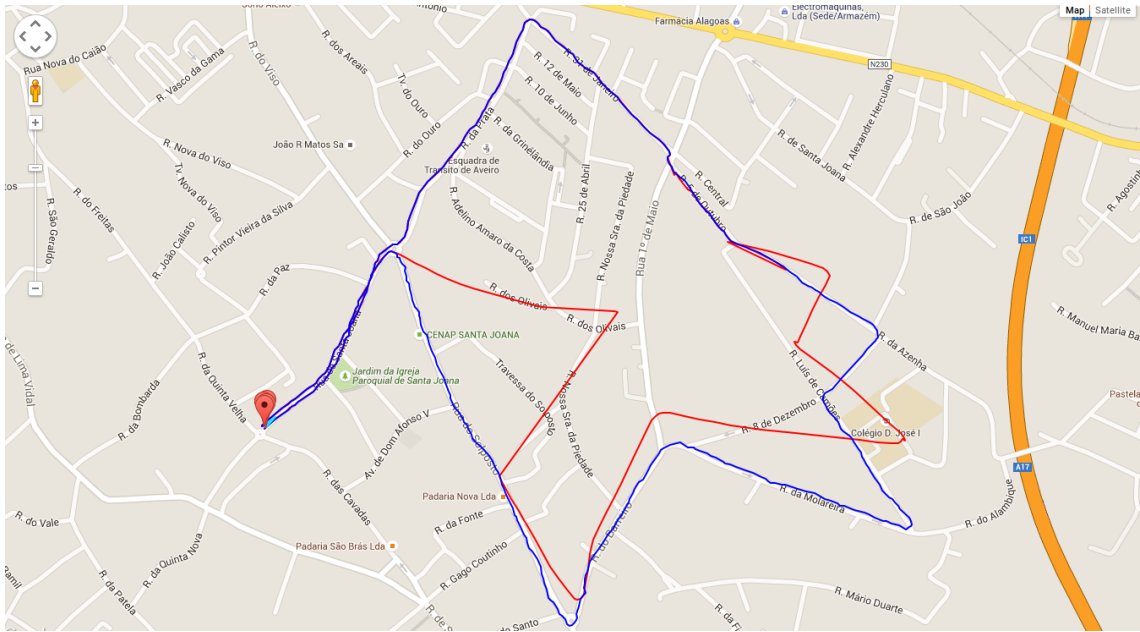


Figure 4.16: Representation in Google Maps of the first run of track 2 using gyroscope-based heading tracker.

Second run In the second run (see tables 4.8 and 4.9), there were GPS signal cuts made between intervals [200, 1200] meters, [1400, 2700] meters, [3000, 4800] meters. This time, contrarily to the first run, gyroscope was more accurate than the magnetometer, with very interesting results, having into account the cut interval sizes.

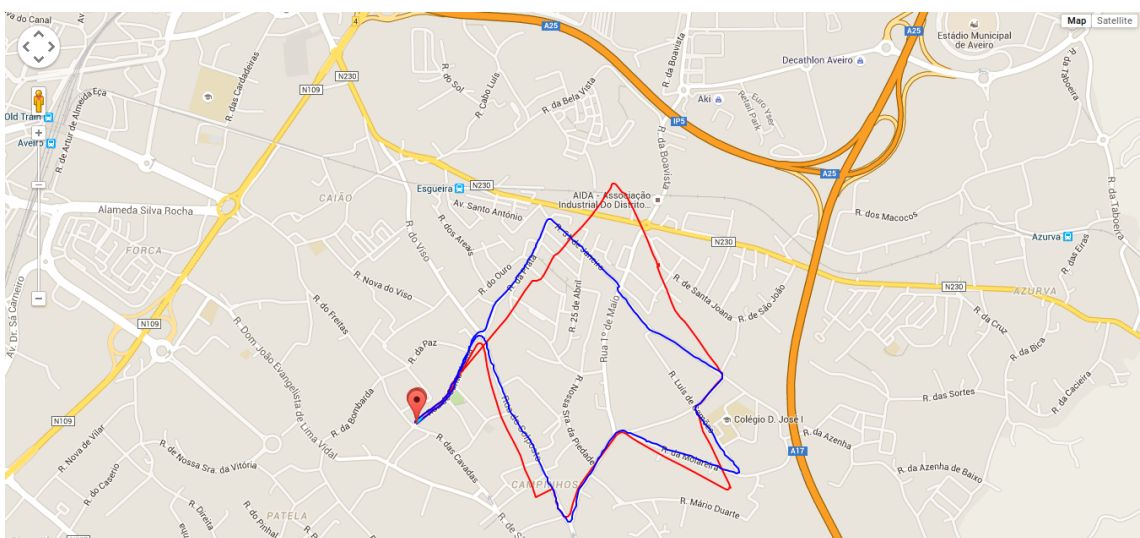


Figure 4.17: Representation in Google Maps of the second run of track 2 using magnetometer-based heading tracker.

Data Collection and Results

Table 4.10: Cuts made to the GPS signal in the second run of track 2, with magnetometer-based heading.

Start	End	Distance	Error	Error per 100 meters
205.6	1200.06	994.46	80.8	8.1250125696
1405.96	2704.21	1298.25	71.31	5.4927787406
3007.2	4806.74	1799.54	314.55	17.4794669749

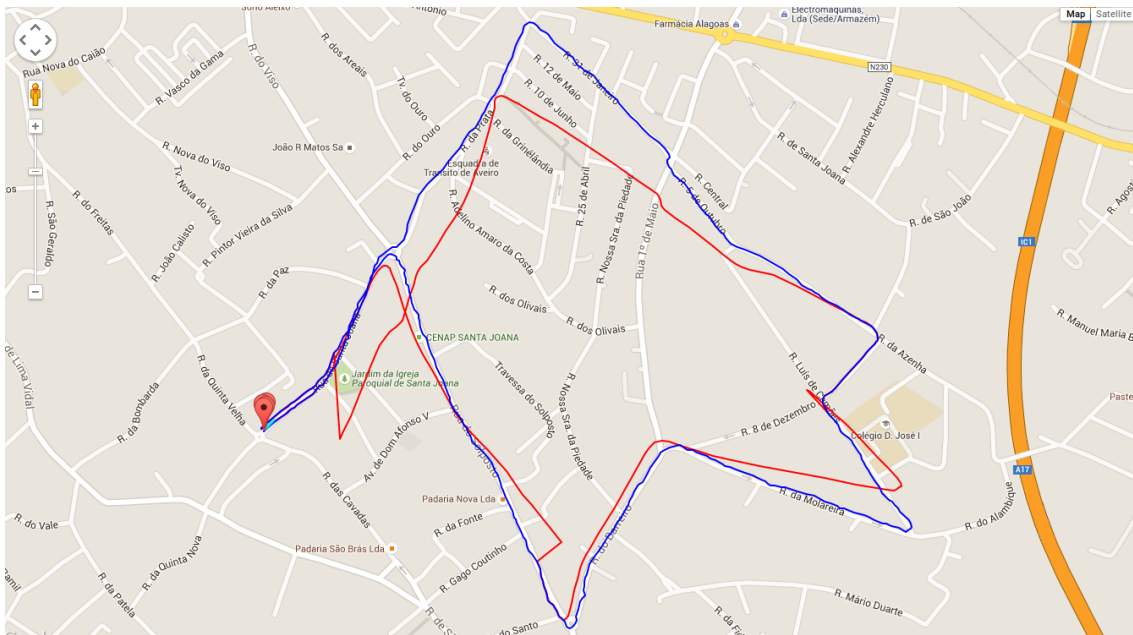


Figure 4.18: Representation in Google Maps of the second run of track 2 using gyroscope-based heading tracker.

Table 4.11: Cuts made to the GPS signal in the second run of track 2, with gyroscope-based heading.

Start	End	Distance	Error	Error per 100 meters
205.59	1200.06	994.47	65.94	6.6306675918
1404.05	2701	1296.95	63.8	4.9192335865
3018.26	4804.13	1785.87	185.48	10.385974343

Chapter 5

Conclusion

After finishing the research in the scope of this dissertation, it is now time to summarize some central conclusions reached throughout the process and establish new goals towards future developments.

5.1 Final Remarks

After analysing some possible approaches to solve the problem presented in the first chapter, it was crucial to engineer a solution feasible in the time frame of this dissertation, but that fulfilled the needed requirements. Such solution was to create a modular inertial navigation system which could use data from the smartphone sensors - the accelerometer, the gyroscope, the magnetometer and the barometer, use an external odometer, connected through Bluetooth Low Energy and use the data provided by these sensors to calculate, using dead-reckoning, approximate positions of the bicycle when GPS signal was not available.

In order to make this system usable in the Syndromic Surveillance context, the calculated positions should focus mainly in latitude and longitude, which would require the measurement of two quantities in each interval of time: displacement and heading. To make these values compliant to the established requirements, some algorithms were developed on top of the raw calculations.

To test the values outputted from the system, a comprehensive test framework was developed on top of vanilla Java and a dataset of bicycle rides was built for usage with the system. This framework will be useful for future system developments and research, since it will allow faster testing to new features, without having to write tests from scratch.

After running the designed test bed, it can be concluded that the system is able to produce, in significant distance intervals, positions that fulfil the requirements established in chapter 3 in the majority of time: reasonable accuracy (500 m^2 - 1000 m^2 around the true point) and low implementation cost, since all the volunteers in the project already have smartphones, and the cost

of the speed sensors is low. It can also be observed that the most problematic aspect (consequently, the one which will need more improvements) is heading calculation.

5.2 Future Work

Although the current implemented solution can be considered a success, after analysing the final results of the experiences, there are some improvement which can be further studied, implemented, and tested:

- **Improve sensor data and GPS data filtering** As already discussed, the smartphone sensors suffer from very high noises, which contribute to drift accumulation and error incrementation. Implementing more complex filtering techniques, such as the Kalman filter can help reduce this drift accumulation. The same applies to GPS: the estimates provided by GPS are fundamental to the correct behaviour of the system. Some other techniques, especially limiting the number of degrees of freedom limitation when the bike is detected to be in a straight line, can be useful to limit accumulation of drift, especially in gyroscope.
- **Use magnetometer and gyroscope to mutually correct themselves** As seen before, there were segments or routes where gyroscope produced much more accurate heading values than the magnetometer and vice-versa. Using both of them at the same time can help reduce calculation errors: if a sudden change of magnetometer-based heading is detected but the gyroscope does not detect a compliant angular speed, then it can be interpreted as an interference; if the gyroscope drift starts increasing too much, when related to the heading indicated by the magnetometer, then correct the gyroscope. However, this is much more complex to properly implement than one can initially assume, since a significant number of variables must be taken into account and tweaked to obtain proper results (such as sampling interval, heading difference threshold, etc.).
- **Implement backward analysis** Backward analysis, in this context, can be very useful to improve the overall reliability of the system - both GPS and INS. It consists in looking to the last calculated values, and with them, make *a posteriori* corrections, based on those same observations.

References

- Abreu, P. H., Xavier, J., Castro Silva, D., Reis, L. P., & Petry, M. (2014). Using Kalman filters to reduce noise from RFID location system. *The Scientific World Journal*, 2014, 1–10. doi: 10.1155/2014/796279
- Aires, K. R. T., Santana, a. M., & Medeiros, a. a. D. (2008). Optical flow using color information: preliminary results. *Proceedings of the 2008 ACM symposium on Applied computing*, 1607–1611. Retrieved from <http://portal.acm.org/citation.cfm?id=1364064> doi: 10.1145/1363686.1364064
- Bidikar, B., Rao, G. S., Ganesh, L., & Kumar, M. S. (2014). Satellite Clock Error and Orbital Solution Error Estimation for Precise Navigation Applications. , 2014(February), 22–26.
- Bitsch Link, J. A., Gerdsmeyer, F., Smith, P., & Wehrle, K. (2012, November). Indoor navigation on wheels (and on foot) using smartphones. In *Indoor positioning and indoor navigation (ipin), 2012 international conference on* (pp. 1–10). doi: 10.1109/IPIN.2012.6418931
- Constandache, I., Choudhury, R. R., & Rhee, I. (2010, March). Towards Mobile Phone Localization without War-Driving. In *Infocom, 2010 proceedings ieee* (pp. 1–9). doi: 10.1109/INFCOM.2010.5462058
- Cook, B., Buckberry, G., Scowcroft, I., Mitchell, J., & Allen, T. (2005). Indoor Location Using Trilateration Characteristics. (1), 2–5. Retrieved from <http://discovery.ucl.ac.uk/136687/>
- Drolet, L., Michaud, F., & Cote, J. (2000). Adaptable sensor fusion using multiple Kalman filters. *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*, 2, 1434–1439 vol.2. doi: 10.1109/IROS.2000.893222
- Facts, G. B. (2000). 2. GPS and GLONASS—Basic Facts. *North*, 13–26.
- Gade, K. (2009). Introduction to inertial navigation and kalman filtering. *IAIN World Congress*(October). Retrieved from <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Introduction+to+Inertial+Navigation+and+Kalman+Filtering#6>
- Galileo Fact Sheet. (2013). , 14–15.
- Integration Drift. (n.d.). Retrieved 2015-01-03, from http://rotations.berkeley.edu/?page_id=1906
- Ivis, F. (2006). Calculating geographic distance: Concepts and methods. *Proceedings of the 19th annual NorthEast SAS Users ...*, 1–10. Retrieved from

References

- <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Calculating+Geographic+Distance++Concepts+and+Methods#0>
- Jackson, G., & Crocker, C. (n.d.). *The use of altimeters in height measurement*. Retrieved 2015-01-08, from <http://www.hills-database.co.uk/altim.html>
- Link, J. A. B., Smith, P., Viol, N., & Wehrle, K. (2011, September). FootPath: Accurate map-based indoor navigation using smartphones. In *Indoor positioning and indoor navigation (ipin), 2011 international conference on* (pp. 1–8). doi: 10.1109/IPIN.2011.6071934
- Merwe, R. V. D. (2004). Sigma-point Kalman filters for probabilistic inference in dynamic state-space models. *PhD thesis*(April), 378. Retrieved from <http://speech.bme.ogi.edu/publications/ps/merwe04.pdf>~~\$\delimiter"026E30F\$~~<http://www.cslu.ogi.edu/publications/ps/merwe04.pdf>
- Muralidharan, K., Khan, A. J., & Misra, A. (2014). Barometric Phone Sensors – More Hype Than Hope !
- Murphy, R. (2000). *AI Robotics*. Retrieved from [http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AI+robotics#5\\$\delimiter"026E30F\\$](http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AI+robotics#5$\delimiter)~~\$~~<http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AI+Robotics#5>
- Pierlot, V., Urbin-Choffray, M., & Van Droogenbroeck, M. (2011). A new three object triangulation algorithm based on the power center of three circles. *Communications in Computer and Information Science, 161 CCIS*, 248–262. doi: 10.1007/978-3-642-21975-7_22
- Sankaran, K., Zhu, M., Guo, X. F., Ananda, A. L., Chan, M. C., & Peh, L.-S. (2014). Using Mobile Phone Barometer for Low-power Transportation Context Detection. In *Proceedings of the 12th acm conference on embedded network sensor systems* (pp. 191–205). New York, NY, USA: ACM. Retrieved from <http://doi.acm.org/10.1145/2668332.2668343> doi: 10.1145/2668332.2668343
- Snyder, J. P. (1987). Map Projections: A Working Manual. *U.S. Geological Survey Professional Paper 1395*, 154–163. doi: 10.2307/1774978
- Vincenty, T. (1975). Direct and Inverse Solutions of Geodesics on the Ellipsoid with application of nested equations. *Survey Review, 33*(176), 88–93. Retrieved from <papers2://publication/uuid/2C837255-3159-4377-B6F2-C5F6D94B3C26> doi: 10.1179/sre.1975.23.176.88
- Woodman, O. J. (2007). *An introduction to inertial navigation*.
- Xavier, J. (2011). *Aplicação do Filtro de Kalman na correcção de dados provenientes de um sistema de Localização baseado em RFID*.
- Xiang, X. X. X., Peng, Y. P. Y., & Zhang, L. Z. L. (2009). A method of optical flow computation based on LUV color space. *Test and Measurement, 2009. ICTM '09. International Conference on*, 2(1), 378–381. doi: 10.1109/ICTM.2009.5413027
- Zaman, S., Slany, W., & Steinbauer, G. (2011, April). ROS-based mapping, localization and autonomous navigation using a Pioneer 3-DX robot and their relevant issues. In *Electronics, communications and photonics conference (siepc), 2011 saudi international* (pp. 1–5). doi:

References

10.1109/SIECPC.2011.5876943

References

Appendix A

Syndromic Surveillance Flyer



Fraunhofer
PORTUGAL



PUBLIC HEALTH
Epidemiologic Surveillance

EPIDEMIOLOGIC SURVEILLANCE

EPIDEMIOLOGIC SURVEILLANCE PLATFORM

Abstract

Surveillance of outbreaks and epidemics by crossing satellite observation data with clinical data collected using mobile devices with lack of continuous network connectivity, on isolated regions of developing countries.

Goals

The project led by Critical Software aims to equip community health workers with mobile devices to collect structured clinical data on underserved populations and cross it with geolocation and earth observation data. Applying methods of business intelligence and through the analysis of correlated data, the solution will allow detecting, monitoring, predicting outbreaks and epidemics and acting to minimize the consequences of infectious diseases such as Malaria and HIV/AIDS.

Mobile front-end module

Fraunhofer AICOS collaborates in this project by providing its PostboxWeb framework that enables offline-capable mobile applications to collect and synchronize data for occasionally connected Android smartphones. PostboxWeb collects data in locations where there is no network coverage and transmit them whenever network is available.

Mobile applications prototypes are developed having a set of front-ends with dedicated interfaces aiming at the massive use of a channel for health records screening, and also featuring the automatic inference of the geographic locations where the clinical information is gathered.

AICOS will also contribute with the technical and scientific knowledge in its areas of expertise, namely:

Contact

Rua Alfredo Allen, 455
4200-135 Porto, Portugal

+351 220 430 300
info@fraunhofer.pt
www.fraunhofer.pt

Industry Client

Critical Software



Clinical Partner

CINTESIS – Center for research in health technologies and information systems

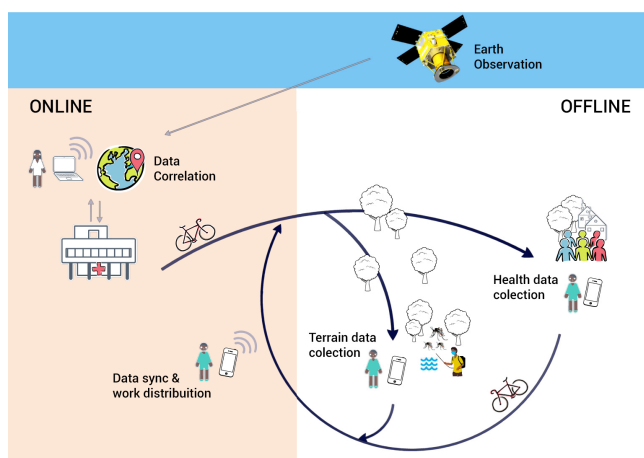


Fig2. Scenarios of Epidemiologic Surveillance Platform.

Mobile front-end module

- Health data
- Terrain data
- Geolocated data
- Secure data storage of clinical sensitive data
- Multiple community health workers sharing a device

Prevent, treat and monitor

- Malaria
- HIV/AIDS

- Information and Communication Technologies for Development;
- Mobile Solutions;
- Human-Computer Interaction.

Interoperability

Epidemiologic Surveillance Platform also aims the development of an interoperable health care monitoring system prototype to allow the surveillance of the infectious diseases, generating estimates of the HIV/AIDS and Malaria epidemic in a given country.

Management module

The platform includes a Health Management module prototype that must be able to receive relevant health information for HIV/AIDS and Malaria surveillance, which will be transmitted by the PostboxWeb. The Health Management module will interact with Electronic Health Records to process

the received data generating statistical metrics to ensure accountability, as well as to monitor the population response to specific therapeutic or preventive programs in demarked regions.



Fig3. End-users of the mobile applications.

Syndromic Surveillance Flyer